RECOMMENDING SERVICES IN A DIFFERENTIATED TRUST-BASED DECENTRALIZED USER MODELING SYSTEM

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

SABRINA NUSRAT

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Masters degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan (S7N 5C9)

ABSTRACT

Trust and reputation mechanisms are often used in peer-to-peer networks, multi-agent systems and online communities for trust-based interactions among the users. Trust values are used to differentiate among members of the community as well as to recommend service providers. Although different users have different needs and expectations in different aspects of the service providers, traditional trust-based models do not use trust values on neighbors for judging different aspects of service providers. In this thesis, I use multi-faceted trust models for users connected in a network who are looking for suitable service providers according to their preferences. Each user has two sets of trust values: i) trust in different aspects of the quality of service providers, ii) trust in recommendations provided for these aspects. These trust models are used in a decentralized user modeling system where agents (representing users) have different preference weights in different criteria of service providers. My approach helps agents by recommending the best possible service provider for each agent according to its preferences. The approach is evaluated by conducting simulation on both small and large social networks. The results of the experiments illustrate that agents find better matches or more suitable service providers for themselves using my trust-based recommender system without the help of any central server. To the best of my knowledge this is the first system that uses multi-faceted trust values both in the qualities of service-providers and in other users' ability to evaluate these qualities of service providers in a decentralized user modeling system.

ACKNOWLEDGMENTS

It is a pleasure to thank those who made this thesis possible. I owe my deepest gratitude to my supervisor, Dr. Julita Vassileva for her continuous support and guidance throughout my work. She has been a great mentor and a tremendous advisor in different phases of my graduate life. I would also like to show my gratitude to my committee members, Dr. Ralph Deters and Dr. Winfried Grassmann, for giving their valuable time and providing constructive comments and suggestions for my work. I would like to thank Dr. Kalyani Premkumar for accepting to be the external examiner of my thesis defense.

I would like to show my gratitude to my parents for their love and support. Without their help this work would not have been possible. I am also grateful to some of my friends and colleagues for their enormous support and encouragement. Last, but not the least, special thanks to all health professionals I contacted during my research, and to faculty and staff members of the department of Computer Science for their support and generosity.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In small towns, people know each other either through personal encounter, reputation of a person in the community, or word of mouth. They form tightly connected networks and information spreads along these networks as rumors and gossip, thus helping people to quickly differentiate someone who is trustworthy from one who is a "bad apple". In large cities, however, anonymity and lack of dense social networks prevent information flow. People need to rely on reputation services, such as the Better Business Bureau (BBB) [3], to recommend good companies, or matchmaking services, to find romantic partners.

People's interactions in virtual environments are similar to that of real-world large cities. They have no mechanisms to allow the spread of information among people about good and bad members. Users are, therefore, exposed to significant risks in their interactions, since they can encounter unreliable, malicious or dishonest users.

Trust and reputation has emerged as suitable mechanisms to ensure that such information is kept and distributed appropriately in the network. Here, trust is one person's belief in another person's honesty, capability and reliability based on direct interaction while reputation, on the other hand, is the collective measure of trust. For example, in a file sharing network, some file providers may be honest in providing files of good quality, while some others are not. Usually, if the user did not have direct interaction with that file provider in the past, it would look for references or recommendations from his friends which indicate their trust in that file provider. After receiving these recommendations, one can aggregate the trust values and make decision whether the file provider is trustworthy or not. This collective measure of trust is called "reputation" and it is used to differentiate good members of the network from bad ones. In the context of selecting service providers, a user looks for someone who is highly trusted by his/her

friends to provide high quality services. Again, people who are giving recommendations (i.e., referees) may not be reliable in sharing information. The user will put more weight on the opinions of his more trustworthy friends than that of others. Trust and reputation mechanisms help users filter information and find trustworthy friends and service providers.

One important feature of trust is that it is multi-faceted, i.e., it has multiple aspects. For example, Wang and Vassileva [47] considered three aspects of file providers (capabilities in providing files of good quality or download speed or different file types) to develop a trust model in a file-sharing P2P networks. Users in such systems need to have differentiated trust in the file provider in multiple aspects. Also, in the case of asking for recommendation, users need to consider multiple aspects of trust in the referees. For example, one can trust his friend's judgment about a paper's readability but may not trust his judgment about the paper's contribution. Therefore, it is important to have differentiated trust model in the referee according to the user's preference criteria (here, readability, contribution). In this thesis, I am presenting a multi-faceted trust model where each user has two sets of trust values: i) trust in the quality of service providers, and ii) trust in the recommendations provided by other users. Some users may be more interested in one aspect than in others, therefore, giving more preference weights in that aspect. Users' preferences are considered in order to find the best possible service providers according to the users' needs. Modeling users' preferences is a problem addressed in the area of user modeling which is presented briefly below.

A user modeling system aims to represent and reason about the goals, preferences, behaviors and information needs of human users [22]. In a centralized system, user modeling servers collect and aggregate user data from many applications so that it can be used for different purposes. However, in decentralized user modeling systems, user models are maintained locally

by the agents [44]. By the term “agents”, I mean the active members of a system/network or software components or web services. They personalize their interaction with users, preserving the context and purpose of the model. In a decentralized system agents don’t need to disclose their personal information to a central server. However, they may need to communicate with each other for sharing information or finding resources/experts in order to pursue their users’ goals.

In a large distributed system, agents can be equipped with user models that represent human users’ preferences and goals. In this thesis, I consider a decentralized user modeling system where users are looking for service providers according to their preferences. Information about a user’s preferences is referred to as “preference model”. In addition, two sets of differentiated trust values are incorporated with detailed user models to find better matches for agents, while preserving privacy. Each agent has a preference model for its user and it uses trust mechanisms for sharing information to find the most suitable service provider for its user. I also present simulation results that have been conducted on social networks. The results of the experiments show that with growing number of requests in the system, users learn from their friends about service providers that closely match their expectation and preferences.

The thesis is organized as follows: chapter 2 surveys the literature in the field of trust and reputation mechanisms and user modeling systems. Chapter 3 explains the problem that is addressed, along with an example scenario and my proposed approach. Chapter 4 describes the simulation design and chapter 5 discusses the experimental settings and results of the simulation. Chapter 6 concludes the thesis with possible directions for future works.

CHAPTER 2
LITERATURE REVIEW

The problem addressed in this thesis and the proposed approach necessitates the review of work done in three related areas: user modeling, recommender systems and trust/reputation models (Fig. 2-1). In this chapter, I present a review of related approaches for discovery of good resources employed by researchers in the field of user modeling and trust and reputation mechanisms. Section 2.1 discusses the application of user modeling in recommender systems and section 2.2 surveys the literature on of trust and reputation mechanisms.

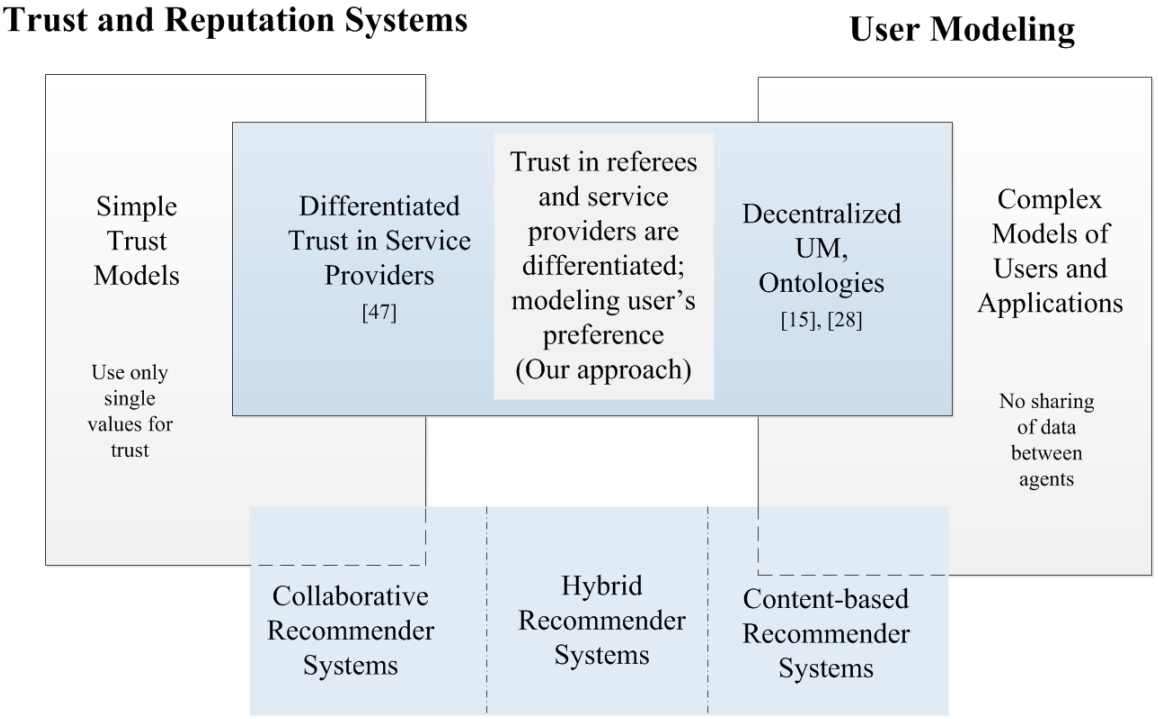

Figure 2-1: Bridging trust systems with user modeling

2.1 User Modeling and Recommender Systems

Section 2.1.1 surveys the literature of user modeling, its classifications and sections 2.2.2 presents a brief review on recommender systems.

2.1.1 User Modeling

User modeling is needed to enable machines to adapt their functions and interface to the human users according to the users' needs and preferences. It is evident that machine models of the users can acquire such ability on a large scale only if they build assumptions through empirical evidences of users' background knowledge, their goals and plans in consulting the system, and their preferences, misconceptions and attitudes [22]. It is thus both conceptually appealing and practically motivating to investigate how such assumptions can be automatically created, represented and exploited by the system during the course of an interaction with the user. However, the requirements of behavioral assumption and interactive update of user models imposes two-fold challenges for the implementation of a user model. These include the problem of inferring knowledge about the user and the procedure of updating the information through user interaction.

There has been a rich collection of research on both domain-specific and generic user modeling systems [43], [21]. Many different approaches to building user models as well as classifications of user models in various dimensions have been proposed [12], [34], [35]. The first classification, proposed by Elaine Rich [35] in 1983, suggests a classification of user models in a 3-dimensional space:

1. Stand-alone vs. collective models,
2. Explicit vs. implicit,
3. Long-term vs. short-term models.

Stand-alone models address a model of a single canonical (typical) user while collective models deal with a collection of models of individual users. In explicit models it is the user's responsibility to create her environment whereas in implicit models the system does it automatically. Finally, user models are also categorized into short-term and long-term models

based on the amount and nature of data and the duration for which it is saved. However, this classification of user models is too generic; therefore some subspaces of these classes have also been defined. Sleeman [39] describes the sub-space of an individual user, implicit, long-term model as the space of student model in intelligent tutoring system. Vassileva [43] proposes two "technical" dimensions in student models that deal with the questions of "what the model represents" and "how the model is created and updated" and introduced classifications in these two dimensions.

There had also been specific user models proposed [20], [45] to address different assumptions on user modeling. For example, Bayesian belief networks have been applied in user modeling problems that are dominated by uncertainty [45]. These models have been effective in building probabilistic models by diagnosing a user's needs and modeling the beliefs, intentions, goals and needs of the user. In an overlay model, the user's knowledge is encoded as a subset of the domain knowledge representation of the systems [43].

It is worth mentioning that the commonly found user modeling systems use a centralized approach. In centralized user modeling architectures, information about the user (e.g. preferences, skills, etc.), and features of the available experts are stored and matched by a central server [12], [20]. User modeling servers are centralized software components that offer their services to several applications in parallel. They allow the integration of existing information sources with information stored in user models for personalized services (Fig. 2-2). Fink and Kobsa [12] discuss some commercial systems that use centralized approach of user modeling.

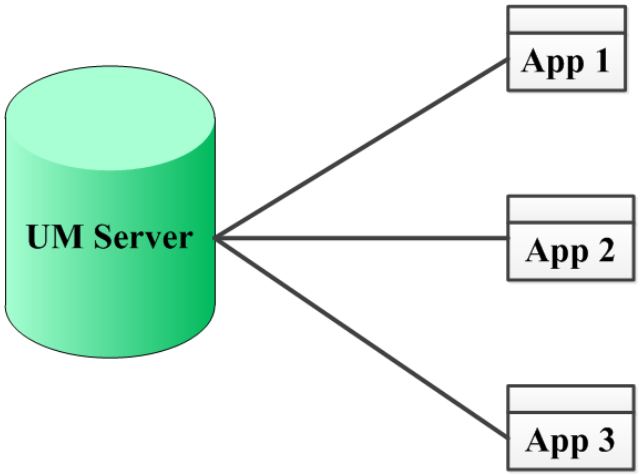

Figure 2-2: Centralized approach of user modeling

In decentralized user modeling architectures, on the other hand, user and expert information is stored by autonomous applications (Fig. 2-3), and is shared and combined by these applications to make recommendation decisions depending on the purpose and context at hand [15], [28].

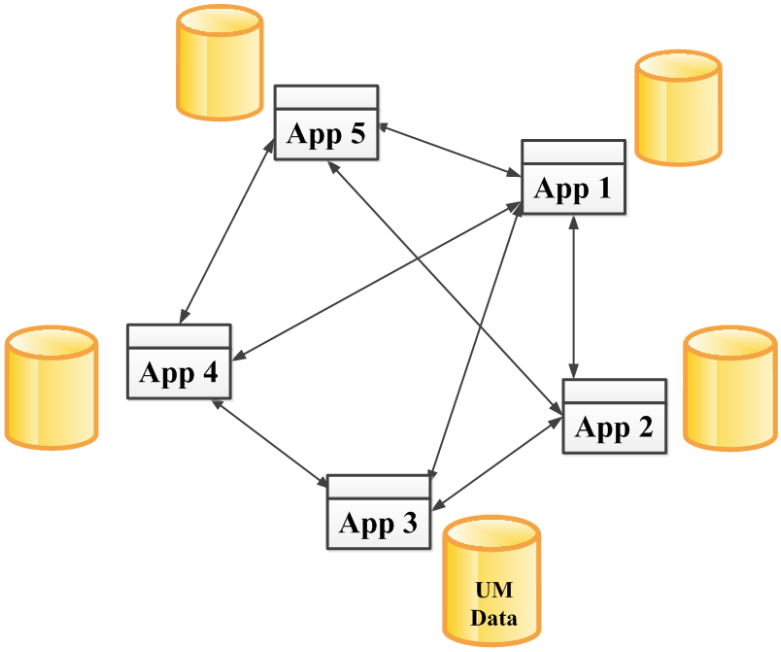

Figure 2-3: Decentralized approach of user modeling

Since the user models are distributed across different applications in a decentralized system, it's important to have interoperability of these models. Ontologies provide a way for representing

and sharing user model data [14], [15]. For example, user modeling markup language (UserML) enables communication between different user models via the Internet [14]. However, these approaches generally assume that the applications that share user information trust each other completely. In reality, some applications may not be trustworthy, and some may provide user data with different accuracy.

2.1.2 Recommender Systems and User Modeling

Since users have different needs in different contexts, an expert finding system should find the right experts according to the users' need. Two main motives for people to look for experts are information needs and expertise needs [52]. In the former case the user only needs to specify his information need while in the latter a more complex specification is required where one is interested in questions like "How much (well) does y know about topic x?" Expert finding systems need to know the expertise levels of other agents as well as the preference criteria of users. Thus practical requirements demand a user modeling system that represents the needs, preferences and aptitudes of users. The model of human user can be managed by a personal agent [44] that stores information about the user's goals, knowledge resources/competencies on certain topics or tasks and about the relationships existing between the user and other users.

Both service selection and expert finding are tightly related to the area of recommender systems. In fact, user modeling was first applied in a system that recommended books to the users [34]. The problem of generating a recommendation for a specific user can be seen as the problem of estimating the user's ratings of the items that have not yet been seen by the user [2]. The ratings given by users capture implicitly the preferences and tastes of users. Ratings of items by a user can be used in two different ways:

1) To develop a model of the preferences and interests of the user with respect to certain features of the items he rated and to apply this model to predict whether the user will like a new

item. This method is used in *content-based* recommender systems, where the users are recommended items similar to the ones they preferred in the past [38]. In order to define similarity between items, typically features of items need to be defined for comparison, and machine learning techniques are deployed.

2) To compare the history of ratings of the user with the ratings of other users. A history of giving similar ratings of the same items means similarity in the users' underlying tastes or preferences. Therefore, the estimation is usually based on correlating the ratings given by this user to items in the past with the ratings of other users who have rated the same items. Once the estimated ratings of unrated items are calculated, the system will recommend to the user the item(s) with the highest estimated rating(s). This type of recommender systems is called *collaborative* [38]. Collaborative recommender systems do not attempt to model explicitly the features of items, relying on strong correlations between rating histories of users, but they require a large set of commonly rated items to calculate a general similarity between users. This similarity can be considered equivalent to the notion of trust between users.

There is a strong correlation between users' trust and service ratings [26]. Users put trust in other users with similar preferences and the ratings of users are also influenced by the ratings of other users that they trust. Therefore, opinions of trusted peers can be used to create recommendations [7]. The users who have had similar service demands in the past can provide useful information for the evaluation of that service provider. However, users may have different satisfaction criteria. If they record and capture subtle details about their experiences, other users can interpret these according to their own criteria and contexts [37]. Thus the recommendation problem can be addressed more efficiently with user models as they can store information about users' needs and preferences.

In this thesis, I use explicit models of user preferences in combination with a trust mechanism in a decentralized environment to recommend service providers to the users. I have given a brief review of the traditional user models in this section. In the next section a closer look is taken at trust and reputation mechanisms.

2.2 Trust and Reputation

Trust and reputation provide "soft security" approaches rather than "hard security" approaches like passwords and authentication to guard against unwanted participants in the system [33]. In the following sections, the characteristics and typology of existing trust and reputation models are explored.

2.2.1 Definition and Characteristics

The study of trust and reputation in multi-agent systems, peer-to-peer networks and online communities has advanced tremendously in recent years. In computer science literature, however, the terms "trust" and "reputation" have been used interchangeably by many researchers. In this work I consider "trust" as a subjective property of a relationship between two peers, evaluated from the point of view of one of the peers. "Reputation" is a collective evaluation of an agent, coming from many agents. It is an aggregate measure of trust from all the agents of a network in a service provider or in an agent. Usually, in a situation where an agent never had any interaction with a service provider, it asks for recommendations from other agents of the network. It combines the recommendations, thus calculating locally the reputation of the service provider. If the reputation exceeds some threshold, it chooses to interact with that service provider. After interaction, it updates its trust value in that provider as well as trust in the agents giving recommendation for that provider [46].

Trust and reputation share some common characteristics [46]. For example, they are *context-specific*. One agent can be considered trustworthy in one context and untrustworthy in another

one. Someone trusted as a doctor may be bad in driving, so he is not trusted as a driver. Therefore, in order to calculate trust one has to select the context, i.e. trust can be calculated in one or another context (disjoint). Trust is *dynamic*; it can increase or decrease with time. It also decays with time. Trust is also *multi-faceted*, it has many aspects. In a peer-to-peer network an agent may consider different aspects of trust in file providers according to its own preferences, for example, providing files of high quality or good download speed etc. For each aspect, it develops trust. The total trust in the file provider will depend on the aggregation of the trust in all these aspects. Therefore, even in the same context, there may be multiple aspects to calculate total trust in a peer. A *multi-faceted* trust model is needed to develop differentiated trust in different aspects of the peer in a given context. Context-specificity can be considered as a disjoint property of trust, whereas the characteristic *multi-faceted* has a cumulative nature. In any context, the total trust in a peer can depend on multiple aspects, thus demanding the use of a multi-faceted trust model.

2.2.2 Typology of Trust and Reputation Approaches

Many studies have been done in the area of trust and reputation and there are several classifications of existing approaches. Sierra and Sabater [36] propose a set of aspects to classify computational trust and reputation models. According to their classification, conceptual models of trust and reputation can be either cognitive or game-theoretical. In the cognitive approach, mental states of the agents are considered. In the game-theoretical approach, trust and reputation do not depend on the mental states of agents, rather on the utility functions and past interactions with other agents.

Traditional models consider two kinds of information sources for calculating trust and reputation: direct experiences and witness information. A large number of models use these sources and consider direct experience to be the most reliable information source for calculating

trust [36]. In some complex models of trust and reputation, however, social relations (e.g. dependence, collaboration, trade etc.) of agents are also taken into account. For example, Carter et al. [4] presented a reputation model where the reputation of an agent depends on the degree of fulfillment of the roles ascribed to it by the society. The authors [4] formalized a set of roles for an information-sharing society and defined methods to calculate the degree of fulfillment of these roles. They proposed five roles on which reputation of an agent depends:

1. Social information provider: 'Users of the society should regularly contribute new knowledge about their friends to the society'.
2. Interactivity role: 'Users are expected to regularly use the system'.
3. Content provider: 'Users should provide the society with knowledge objects that reflect their own areas of expertise'.
4. Administrative feedback role: 'Users are expected to provide feedback information on the quality of the system. These qualities include ease-of-use, speed, stability, and quality of information'.
5. Longevity role: 'Users should be encouraged to maintain a high reputation to promote the longevity of the system'.

The total reputation of an agent will be the weighted aggregation of the degree of fulfillment of each of the above roles. It depends on the specific society how much weight it wants to give on each of these roles. This overall reputation is calculated in a central point and shared among all other agents.

Prejudice is considered to be another source for calculating trust and reputation, though its use is not very common. Prejudice is a mechanism to assign some properties to an agent based on the signs (for example, uniforms) that identify the group it belongs to [36]. Although "prejudice"

refers to a negative or hostile attitude toward another race, gender or social class in human societies, for agent societies, its use does not involve negative connotations.

In some agent societies, where agents can lie or cheat, trust and reputation models need to have mechanisms to deal with liars. Sierra and Sabater [36] categorized three classes of trust reputation models based on agent behavior assumptions:

1. Level 0, where agents are assumed to be truthful and to provide honest rating
2. Level 1, where agents' behavior can be biased although they do not lie
3. Level 2, where agents can cheat and models need to have specific mechanisms to deal with it

Wang and Vassileva [49] analyze trust and reputation models from a different perspective and divide them into two distinct classes: centralized vs. decentralized. In a centralized approach, there is a central server which stores the trust values of all agents and calculates reputation. In decentralized systems, on the other hand, agents are responsible for sharing information and calculating reputation of other agents. Both centralized and decentralized systems can be categorized into two distinct classes: person/agent vs. resource. In person/agent systems, the emphasis is on building reputation of users of the system. Resource systems, on the other hand, work on building reputation of resources. The authors [49] added a third level hierarchy to classify the trust and reputation models: global vs. personalized. In global systems, reputation is based on the public opinion and it's visible to the whole population. In personalized systems, on the other hand, reputation of an entity (person/agent/product/service), for a member, is built on the opinions from a particular group, selected by the particular member. Therefore, for one particular member, the reputation of the entity is personalized.

2.2.3 Review and Analysis of Significant Computational and Online Trust/Reputation Models

Marsh [25] was the first to propose a computational trust model in 1994 in his PhD thesis. It takes into account only direct interaction and differentiates three types of trust: basic trust, general trust and situational trust. Basic trust is the general trusting disposition of an agent independently of any other agent or situation. General trust is the trust one agent has on another agent without taking into account any situation. Situational trust considers trust of an agent on another agent in a specific situation. Trust is represented in his model as a subjective number between +1 and -1. Several limitations exist in this simple model - it does not consider witness information and based its calculation only on direct experiences. Abdul-Rahman et al. [1] has differentiated between direct trust and recommender trust. According to their model, one agent can have direct trust on another agent and this trust has only four distinct values ("very trustworthy", "trustworthy", "untrustworthy", and "very untrustworthy"). The other trust comes from recommendations, which they call "reputation" [1].This is a "discrete trust model" that uses discrete values for trust rather than continuous measures [17].

Mui, Mohtashemi and Halberstadt [27] describe a model that reinforces the relationships among trust, reputation and reciprocity. Increase in one agent's reputation in its embedded social network should also increase the trust from the other agents for this particular agent. Again, an increase in an agent X's trust for Y should also increase the likelihood that X will reciprocate positively to Y's action. This form of reciprocity can also be seen in online transactions. For example, in eBay, users can give three possible ratings after the completion of a transaction: positive (1), negative (-1) or neutral (0) [33]. The reputation value is computed as the sum of these ratings over the last six months. There is a high correlation between buyer and seller feedback. Buyers strongly depend on the reputation of sellers. Sellers try to avoid negative

feedback. Once someone has a black mark by one's name, others may be more willing to cast stones on him. Given the same quality of service, people will be more critical about him. This is a process called "stoning" [33] which swiftly turns the system against undesirable sellers and helps it work with more trustworthy users. The analysis provided some insight on how sellers build their reputation and how such reputations turn into trust.

Reputation mechanisms are implemented in social networking websites and online marketplaces to emphasize trust-based interactions between users. There are different reputation patterns depending on the type of the online community [6]. In competitive societies, there are numbered (1, 2, 3 etc.) or named (gold, silver, bronze etc.) levels to differentiate between different users, e.g., Yahoo! Music ratings. Rankings are another type of reputation pattern; like levels, they make clear comparisons. They are used in highly competitive societies. A rank is measured in terms of accumulated points. For example, "Yahoo! Answers" awards points to users: 100 points to begin participating on Yahoo! Answers, -5 points to ask a question, 3 points to choose a best answer for one's question and so on. Based on the accumulated points, there are different ranks between users.

A leaderboard is a list that shows the top contributors of the system ranked by score [6]. Leaderboards are used in highly competitive society where the fundamental purpose of the community is competition, like, in fantasy sports or games. However, these kinds of reputation mechanisms make clear comparison between people and are considered cold and impersonal, especially by people who do not like to be graded. In less competitive societies, labels can be used, e.g. "helpful", "elite" or "official". They do not make clear comparison. Yahoo! Answers uses this mechanism and awards a top contributor badge to "someone in the Answers community who, through their participation on Answers, has shown that they are knowledgeable in a

particular category". Awards provide incentive for positive behavior within a community and encourage quality participation [6]. They can be given by the host community or from one peer to another. Awards are useful to recognize first-time achievements (e.g., "first recipe written") and encourage participants of the system to try new and novel features. Another reputation pattern is "Top X", e.g., Amazon's "Top" reviewers. Amazon allows members to write reviews on books. Anybody can sign up to be a member and rate a book in a scale of 1 to 5 stars as well as write a prose text of review. An average of all these ratings makes up the online rating of the book. Users, including non-members, can vote the reviews to be "helpful" or "not helpful". Based on the number of "helpful" votes and some other parameters, not publicly revealed, Amazon determines its top reviewers, e.g., top 10 or top 100 reviewers, etc. [6].

The survey paper by Audun Josang et al. [17] gives an overview of significant trust and reputation models and analyzes the current trends and developments in the area of online reputation systems. In some systems, trust/reputation scores are calculated by repeated loops or iteration. Models used in these systems are called flow models [17]. For example, in the EigenTrust model trust scores are computed by iterative multiplication and aggregation until the trust value for each agent in the P2P network converges to some stable value [18]. Some flow models assume a fixed value for the overall system's reputation and one participant can only increase its reputation at the cost of others.

Models based on Bayesian systems compute trust and reputation scores by statistical updating of beta probability density functions [17]. They take binary ratings (positive or negative) as input. When trust is used in decision making and the distance between information source and sink is significant, it is important to calculate an accurate estimate of trust in the information. Kuter and Goldbeck's trust inference algorithm, SUNNY [24], finds all possible paths from a

source to a sink in social network and uses a probabilistic sampling technique to estimate confidence (or, belief) in the trust information. Trust and reputation models can be built on fuzzy concepts, where they have functions that describe to what degree an agent can be trustworthy or untrustworthy. Belief models work with probabilities, but the summation of probabilities of all outcomes does not necessarily add up to 1; the remaining part is termed as "uncertainty". Yu and Singh [53] have proposed a method based on belief theory to calculate reputation scores. The beliefs considered in this model are that either an agent is trustworthy or untrustworthy. These scores are then combined using Dempster-Shafer theory. This is a generalization of Bayesian probability theory which provides a method for combining beliefs or evidences from different sources. A trust transitivity model proposed by Qie et al. [31] is also based on Dempster-Shafer theory. It describes a way of propagating and combining different types of trust (directness trust and recommendation trust). However, in all the models mentioned above trust is represented as single value (mostly, in the range between 0 and 1) and multiple aspects of trust are not considered.

Agents can form interest-based communities in decentralized systems that help them share knowledge and achieve some common goals. In a decentralized peer-to-peer network, reputation plays an important role in community formation and update [48]. Reputation of an agent in the community determines whether the agent is eligible to be a member of that community or not. As there is no central server in a decentralized society to reward or punish the bad agents, agents can use social mechanisms to get rid of undesirable users. If an agent *X* encounters a bad agent *Y* during some interaction, *X* will decrease its rating of *Y* and inform its neighbors. An agent who receives this information can combine it into its trust model of *Y*. This is how agents can "gossip" and bad agents are penalized [53].

In a file-sharing peer-to-peer network, an agent may consider different aspects of trust in file providers according to its own preferences. A Bayesian network-based trust model by Wang and Vassileva [47] combines trust in different aspects to calculate total trust in service providers. Hang et al. [13] describe propagation of trust by three operators (concatenation, aggregation and selection) and claim that their approach can accommodate multidimensionality of trust. However, their approach doesn't model user's preference over multiple aspects of trust.

2.3 Summary

From the above discussion it is clear that the study of trust and reputation and the study of user modeling have advanced independently. Trust models are typically simple and use only single trust values that one agent holds about another one. Fig. 2-4 shows a network where agents A, B and D are connected in a network and S is the service provider. If traditional approach of trust is used in the network, then the trust value that one agent holds about another can be represented by a single value as illustrated in the figure.

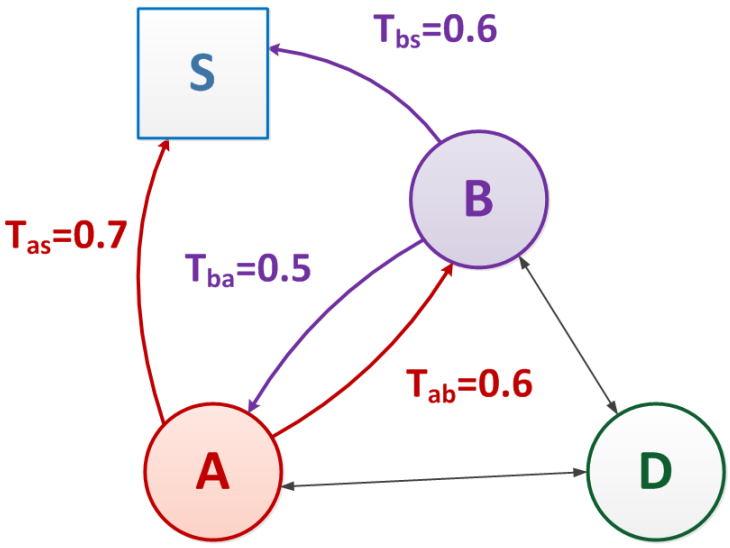

Figure 2-4: Traditional approach of trust

Most of these approaches do not consider other information about users, such as their preference criteria and differentiated trust values on the recommendations received. More

advanced mechanisms require modeling based on users' preferences in multiple aspects. In few models of trust, trust in multiple aspects of service providers is considered, but trust in friends as providers of references is still single-valued. In a decentralized user modeling system, on the other hand, user models store a lot of information about individual agents as there is no central server. However, sharing of information among user models assumes that the agents sharing user models are honest and trusted by default, which is not justified in an open environment where agents may be self-interested and/or malicious. Even if one assumes honesty, agents may have different criteria and use different procedures in building their user models and develop these models in different contexts. So there is no guarantee that the information contained in a model developed by one agent can be trusted by another agent. In the case of sharing information, trust and reputation should play an important role.

2.4 Current Research Directions

It is evident from the discussion that there is a gap between the research areas of trust and reputation mechanisms and decentralized user modeling. Augmenting decentralized user modeling with trust and reputation mechanisms has the potential to improve user modeling and service/ expert recommendation. This is depicted in Fig. 2-1.

In this thesis I present an approach that bridges the approaches of trust models and decentralized user modeling systems. I use *multi-faceted* trust models for developing trust in different aspects of the service providers and also for developing trust in friends to provide fair judgment in these aspects in the context of selecting suitable service providers according to users' preferences. Trust mechanism will be used along with a user modeling approach for sharing information among users. This information will be processed in different ways depending on the preference criteria and trust models that are incorporated with the user models.

This approach is expected to find better matches for the users as it considers users' preferences and differentiated trust on other users.

CHAPTER 3
PROBLEM STATEMENT AND PROPOSED APPROACH

3.1 Problem Statement

I address the problem of finding service providers for users in a decentralized environment. From the discussion in chapter 2, it is evident that the traditional trust models consider single value to indicate trust in each other; they don't consider difference in capability to judge different aspects. Also these models usually do not store users' preferences and needs. In decentralized user modeling systems, on the other hand, a lot of information about users' needs, preferences and goals are stored in autonomous applications. But the sharing between applications is done in a simplistic way, without considering that there might be different trust among applications. I want to bridge these two approaches to find better matches or more suitable service providers for the users.

3.2 Proposed Approach

In a decentralized environment, user models will have information on goals, needs and preferences of individual users. Agents can query each other and find trustworthy service providers. There are two features in the system that I am addressing:

1. Agents will have different preference models. Agent *a1* may put more preference weight on criterion "x" than criterion "y" while looking for a service provider whereas agent *a2* may do the opposite. (Here, by "criterion", I mean a "facet" or "feature" or "aspect" of a service provider)
2. Agents in the decentralized system use trust mechanisms to aggregate the recommendations received from others. However, agents have differentiated trust in both service providers and referees. An agent may judge a service provider s1 as more trustworthy with respect to criterion "y" of the service s1 provides than to

criterion “z”. Also the agent may consider another agent as capable to judge fairly criterion “x” of the service provided by *s1* and thus trust the other agent as a referee with respect to criterion x. However, the same agent may not be as trustworthy in judging criterion “y”. Hence, two types of trust exist in the system: trust in service providers and trust in referees.

Therefore, our proposed approach deals with agents, their preferences and differentiated trust in other agents and service providers to help them find better matches. Each agent has to calculate its trust in a particular service provider based on its own preferences, trust model and the references about service providers, received from other agents. An example scenario and the proposed approach are described below.

3.3 Example Scenario

Consider the situation where patients are looking for doctors based on their preferences. The preferences are subjective (different people have different preferences). While most people are concerned with the competence of the doctor, some consider the doctor’s nationality, gender, or how approachable / friendly the doctor is, as important qualities. In a simple case, three criteria can be considered: the doctor’s expertise level, how approachable the doctor is and his availability. Users have differentiated trust values in doctors they know based on their preference criteria. They also have different trust in their friends to provide accurate judgment about different aspects of doctors.

Consider first a traditional centralized user modeling approach to deal with the situation. A classical centralized user modeling system will have a central server and a set of models of doctors, represented according to a specific schema, e.g. their competencies in a set of different areas, plus some other features, e.g. demographic (gender, nationality), and personality traits (approachable, or not). Users may be represented by models on the server that contain

information about their health needs and preferences. When a new user comes, she will have to fill in a request to join, thus bootstrapping her own user model on the server. The matchmaking application will query the server with respect to the features of all doctors represented on the server and will find a match for the user's request. Normally, there will be no comparison of the user's request with other user's requests, or any explicit communication between the users, sharing trust values, etc. However, there are privacy issues – users have to divulge their information to the central server. The server will need to have mechanisms to protect this data. It also needs to have mechanisms to update the user models and the doctor profiles.

In more advanced matchmaking systems, when a direct match cannot be found, it is possible that the matchmaker correlates user models and recommends a doctor using collaborative recommendation algorithm [2]. Yet again, there is no explicit sharing of data between users that are connected with social links (friends, relatives etc.) or who trust each other. The correlation is implicit and based on their user models, which are voluntarily divulged by users to the server. The users are not even aware that they are compared with other users by the system.

My approach is decentralized, and presents an alternative to the centralized user modeling approach by exploiting the relationships based on trust among user models. The users do not divulge information to a central server. All calculations are done by agents residing on client applications and maintaining the user model of their users and trust models of other users (friends of users). The agents communicate with the agents of other users who have established trust connections among themselves. Fig. 3-1 depicts this scenario.

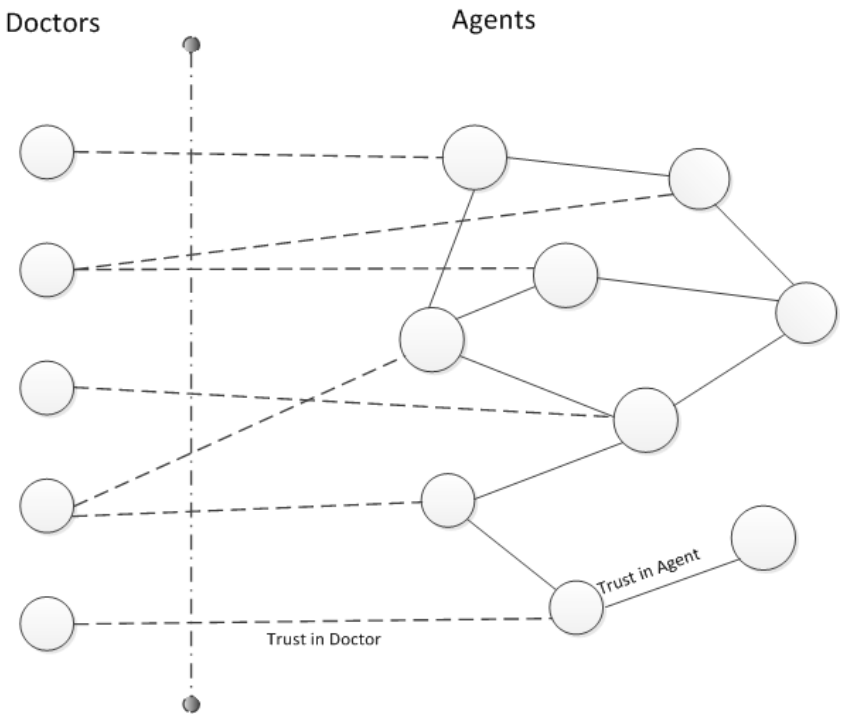

Figure 3-1: Network of agents and doctors

The scenario for finding specialist doctors applies to many countries with market-based health-care system. In Canada, however, patients don't have much freedom to select specialists for themselves. Usually, the only option available for them is to interact with the family physician who would direct them to a specialist. I used this hypothetical scenario of doctor-patient only to explain my model. My approach can be applied in any scenario where users can select service providers or resources for themselves; for example, finding suitable lawyers or nannies, finding books or movies according to the users' preferences etc. The features or preference criteria may also be different in these scenarios depending on the purpose and context.

3.4 Model

I will now give a detailed description of my user modeling and trust approach based on the example of doctor-selection scenario. Consider that two agents *a* and *b* are connected in a

network. Agent *a* is looking for a doctor. It has its preference model P_a which contains its preference criteria for doctors and the corresponding preference weights (see Fig. 3-2). Here, three preference criteria and their weights are considered. These are doctor's *expertise level*, *how approachable the doctor is* and doctor's *availability* denoted by $P_{expertise\_level}$, $P_{friendliness}$ and $P_{availability}$ respectively. For example, in the preference model of agent *a*, it may give weight 0.5 to the preference criteria doctor's *expertise level*, weight 0.3 to *how approachable the doctor is* and 0.2 to the doctor's *availability*. That means, for this agent, $P_{expertise\_level} = 0.5$, $P_{friendliness} = 0.3$, $P_{availability} = 0.2$. However, the agent keeps this information to itself. This means that the preference model of the user is not divulged to a centralized server, but is known only by the user's personal agent. In principle, in a service selection scenario there can be many other preference criteria. This model scales well with the number of criteria.

Agent *a* also has a trust model (denoted by T_{ab}) for agent *b* who will be used as a referee. T_{ab} is a differentiated trust model in *b*'s ability to provide fair judgment on a doctor's qualities. For this example given, T_{ab} consists of $T_{expertise\_level}$, $T_{friendliness}$ and $T_{availability}$ where these represent *a*'s trust in *b*'s judgment on doctor's *expertise level*, on doctor's *friendliness (approachability)* and on doctor's *availability* respectively. Suppose *a* trusts *b*'s judgment on doctor's friendliness and availability, in this case the trust values $T_{friendliness}$ and $T_{availability}$ will be higher. But *a* has doubt about *b*'s judgment on doctor's *expertise level* (may be because *b* has only seen the doctor once); that means, $T_{expertise\_level}$ will be low. Therefore, *a*'s trust model about *b* (T_{ab}) will have differentiated trust values on judgment of *b* for different preference criteria.

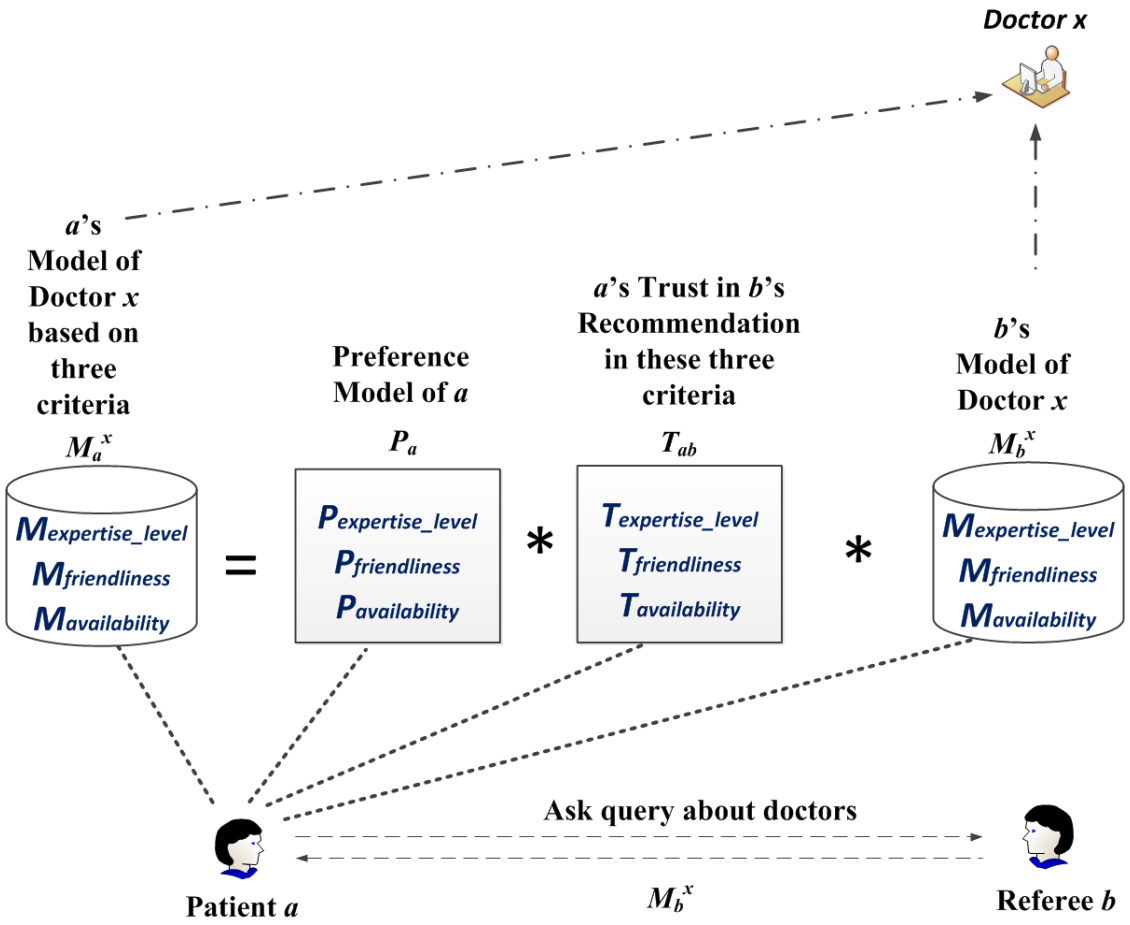

Figure 3-2: Interaction between patient *a* and referee *b* (where $b \epsilon B$)

The set of all agents that could be used as referees by agent *a* is denoted by *B*, where *b* is a member of set *B*. Set *B* is static for any agent *a* since the social graph remains the same while running the simulation of the system.

Agent *b* has its own models for different doctors. Consider a doctor *x* from the set of doctors *X*. For doctor *x*, *b*'s model is M_b^x. This model is *b*'s beliefs of *x*'s features derived by its own experience or by asking others for reference.

The following procedure is executed to request evidence:

1. *a* requests references from *b* (*b* is a member of the set *B* of all the agents *a* knows) about its models of doctors.
2. *b* sends back M_b^x
3. *a* considers the received reference by taking into account its own trust in *b* as a referee, T_{ab}; that is, *a* computes $T_{ab} * M_b^x$
4. a creates its own model for doctor x by taking into account its preference model P_a and all the data received from references belonging from the set B about x.

Taking references from all referee-agents, *a* computes its model of doctor *x*, M_a^x, with this equation:

$$M_a^x = \sum_{b \epsilon B} (P_a * T_{ab} * M_b^x) / ||B|| \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Here, $||B||$ denotes the cardinality of set *B*. "*" denotes element-by-element multiplication of the vectors; that means, *i*-th element of P_a is multiplied by *i*-th element of both T_{ab} and M_b^x and the same goes for other elements of the vectors.

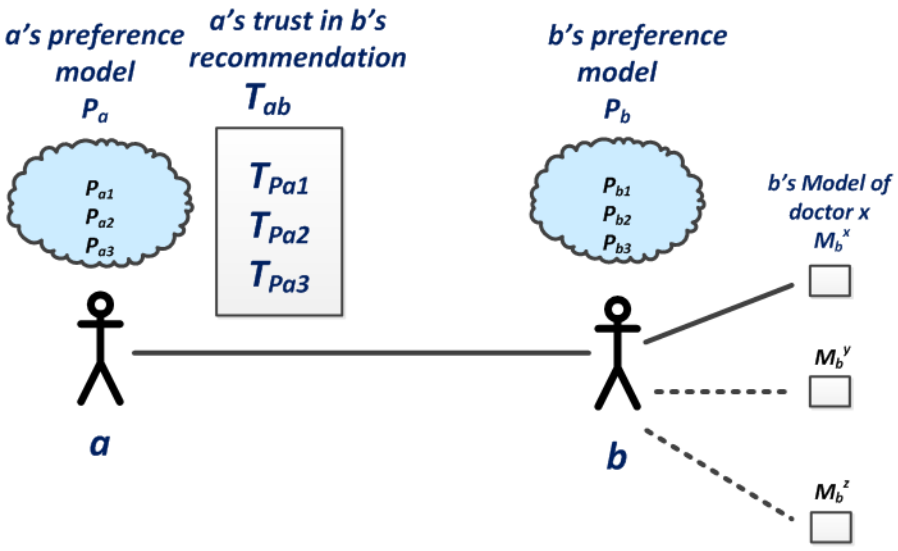

Figure 3-3: Trust and preference models between two agents

The example scenario presented in this section can be generalized and my approach can be applicable in any network where users share information with their friends and rate service providers according to different criteria. Figure 3-3 shows a generalization of my approach where three preference criteria of agent *a* are denoted by P_{a1}, P_{a2} and P_{a3} respectively. Similarly, *a*'s trust in *b*, T_{ab} is a differentiated trust model that consists of three features T_{Pa1}, T_{Pa2} and T_{Pa3}. Here, the criteria P_{a1}, P_{a2} and P_{a3} depend on the scenario the model is being applied on. In reality, there can be more than three criteria that users consider while looking for service providers. Agent *b* can have models of more than one doctor. In Fig. 3-3, agent *b* has models of three doctors, M_b^x, M_b^y and M_b^z respectively.

CHAPTER 4
SIMULATION DESIGN

In order to evaluate my approach, I conducted simulation of my recommender system on two kinds of graphs: a synthetic graph with small-world properties and a large scale social network. The objective and design of the experiment, performance metric and computational steps are described below.

4.1 Objective

The significance of my approach is evaluated by running simulation on social networks. The focus of the experiments is to see how effectively my trust-based recommender system helps an agent find appropriate expert that matches its preferences. It is expected that the average performance of the agents in the system should increase with the number of queries.

4.2 Simulation Language and Data Storage

Erlang [8] is chosen as implementation language of the simulation because it is a functional language that is well-suited to building large-scale distributed systems. The runtime system of Erlang has built-in support for concurrency, distribution and fault-tolerance. Erlang processes communicate using message passing and this feature is used for communication among agents in our simulation.

Erlang dets [9] table is used for storing data. It stores data as tuples and organizes them as a linear hash list, which grows gracefully as more data is inserted into the table. A tuple is a compound data type in Erlang with a fixed number of terms, e.g. for N terms, the tuples consists of

{Term1,...,TermN}

Several dets tables are used to store data in different stages of simulation. The tuples in these tables consist of numbers, atoms (constants) and lists. For example, suppose, one dets table in

the system contains agent's name (atom), referee's name (atom), doctor's name (atom), doctor's model (list), number of references (number). Therefore, the tuple is:

{Agent, Referee, Doctor , Reference_list, Number_of_References}

One instance can be: {a7, a23, d150, [0.5, 0.9, 0.7], 1}

A generic server process (gen_server) [10] is implemented for each of the dets tables to synchronize access and updates to the database.

4.3 Input Parameters

The service providers are characterized with a differentiated model containing several features. In the patient-doctor scenario discussed above, this model consists of three features:

1. Expertise level of the doctor,
2. How approachable the doctor is (or, friendliness) and
3. Availability of the doctor

All of the above parameters can have values ranging from 0.1 to 1.0. If K is a differentiated model of three features of a doctor, then K can be represented by:

$K = \{K_1, K_2, K_3\}$

Here, K_1, K_2 and K_3 respectively denote doctor's level of expertise, friendliness and availability.

Similarly, if P represents a user's preferences with respect to these three features, then

$P = \{P_1, P_2, P_3\}$.

The values for P_1 and P_3 range from 0.1 to 1.0, whereas value of P_2 range from 0.1 to 0.5 only, based on an assumption that patients, while looking for doctors, will give more priority on the doctor's expertise and availability than on the doctor's friendliness. All these values are random float numbers uniformly distributed in the range (0.1 to 0.5 or, 0.1 to 1.0) for all the agents.

Each agent has differentiated trust in other agents that are part of its social network and are therefore potential referees. These trust values indicate how much the user trusts the referee to provide fair judgment with respect to the different features. For our experiment, the trust models included three values that indicate trust in the referee for providing fair judgment in doctor's expertise level, his friendliness and availability respectively. These trust values also range from 0.1 to 1.0.

All these initial values of parameters were generated using Erlang random number generator. The Erlang random number generator is deterministic and seeding it with particular values forces it to produce the same sequence of numbers across runs. I seeded the random number generator with particular numbers at the beginning of each simulation run. This was done in order to have the same initial setup for each run so that the results can be reproducible.

4.4 Performance Metric

The best possible doctor for an agent is determined according to its preferences and a score is calculated representing it. If *K* represents a differentiated model for a doctor *x*'s features and *P* represents agent *a*'s preferences with respect to these features, then score (*A*) of that agent, for this doctor, is calculated as follows:

$A_a^x = \sum_i K_i * P_i$, *where* $i \epsilon \{1, 2, 3\}$ ……..*(2)*

For the example scenario I described in chapter 3, if patient *a*'s preference model is such that it gives weight 0.5 to the preference criteria "doctor's expertise level", weight 0.3 to "how approachable the doctor is" and 0.2 to "doctor's availability", then $P=\{0.5, 0.3, 0.2\}$. Suppose doctor *x*'s current values for these features are such that $K = \{0.6, 0.7, 0.8\}$. Then, according to equation (2), *a*'s score for *x* will be 0.67.

The present scores for all the agents are computed at the beginning of the simulation and the matches with the best possible scores are identified. This is only necessary in the simulation in

order to identify an overall measure of "goodness" of every possible match and thus, to be able to evaluate if agents are improving their ability to find good matches by interacting with their neighbors. Note that in a real system this will not be feasible, since there does not exist a central point that keeps models of the preferences of all users. This computation is done only in order to evaluate our approach.

With each request, the agent that sends a request learns more about doctors from the references it receives from its friends, and finds a better doctor according to its preference. At any point of time, the percentage of satisfaction (*Sat*) for an agent is determined by equation (3).

$$Sat = (A_{current}/A_{best})*100 \quad \ldots\ldots(3)$$

Here, $A_{current}$ represents the current score for present choice of doctor (explained later) and A_{best} represents best score achievable for the patients in the set of all doctors in the system.

An increase of *Sat* indicates that the user is getting closer to the best possible match in the network. As more requests are generated in the system, it will be observed how close the agents get to their best matches using my trust-based recommender system.

4.5 Simulation Algorithm

The experiments were run following some computational steps. Fig. 4-1 summarizes various processing operations involved in my simulation experiment.

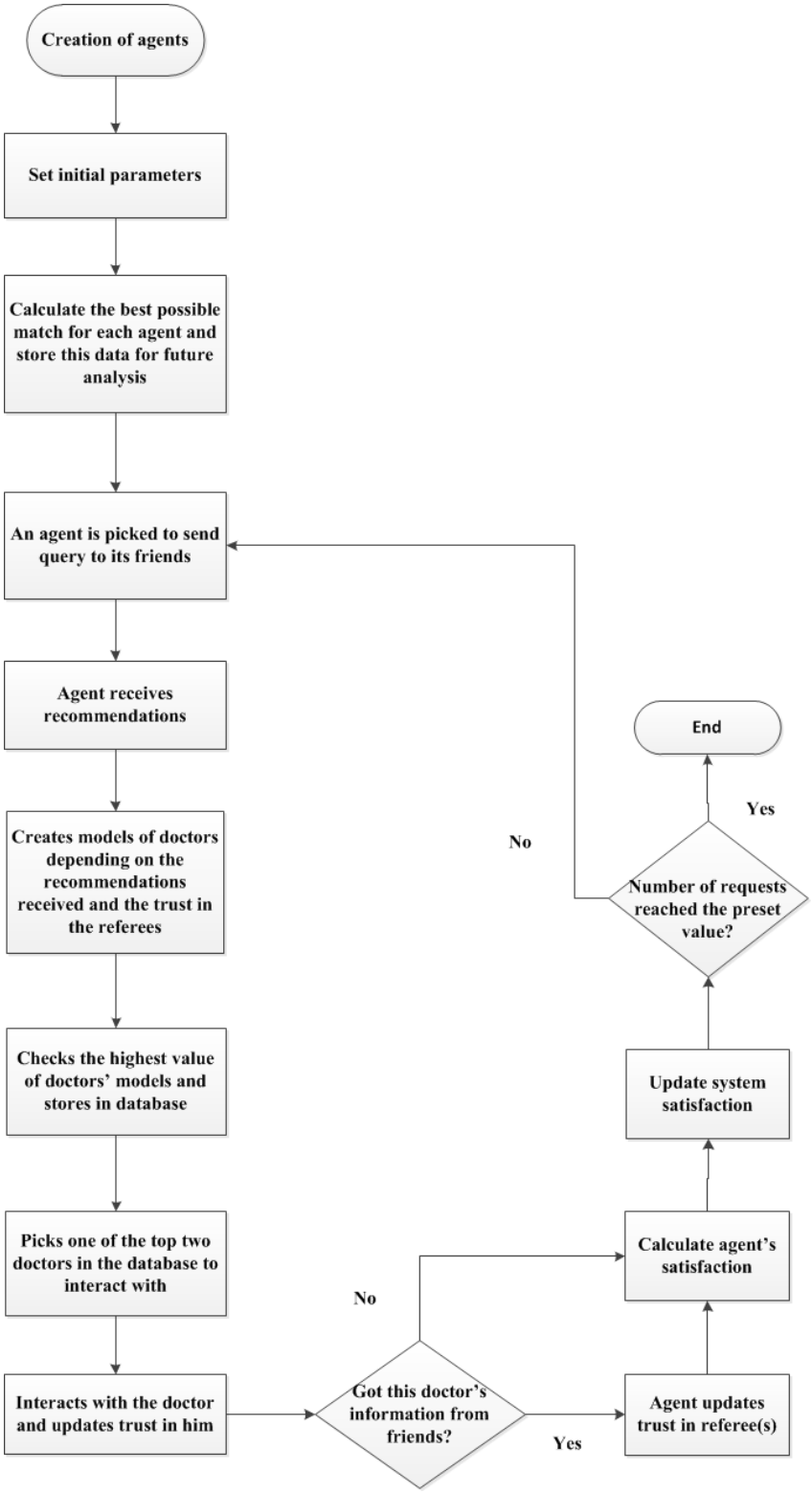

Figure 4-1: Flowchart showing the steps in my approach

After an agent is created, an agent can perform the following activities: sending queries to other agents with whom it is connected (friends in the social network) about service providers (doctors), processing feedback from these referees, creating models of doctors, deciding which doctor to interact with, interacting with the doctor, updating trust in doctor, updating its differentiated trust in its friends as referees and calculating its own satisfaction. After each query is processed, system's average satisfaction is computed which indicates how well agents are finding their matches during the course of simulation using their trust-based network. Each of the execution steps is explained briefly below.

Defining Optimality Criterion: In the first step of simulation, the best match for each of the agents is calculated in the system. This indicates an agent's most optimal choice of service provider (doctor) from the entire pool of service providers. In order to find the best match, all possible scores of the agent are calculated using equation (2). Then the service provider is identified for whom the score is the highest. This indicates the agent's best or most optimal choice. The best matches for all the agents are identified in the beginning of simulation.

Creation of agents: A separate thread of execution (i.e. process in Erlang) is created for each of the agents in the model. These threads (agents) are distributed in different Erlang runtime systems (see Appendix A). These runtime systems are connected with each other so that all the agents can send messages to one another. Fig. 4-2 shows a snapshot of the simulation in a distributed Erlang system where eight Erlang windows represent eight Erlang runtime systems that are able to communicate with each other.

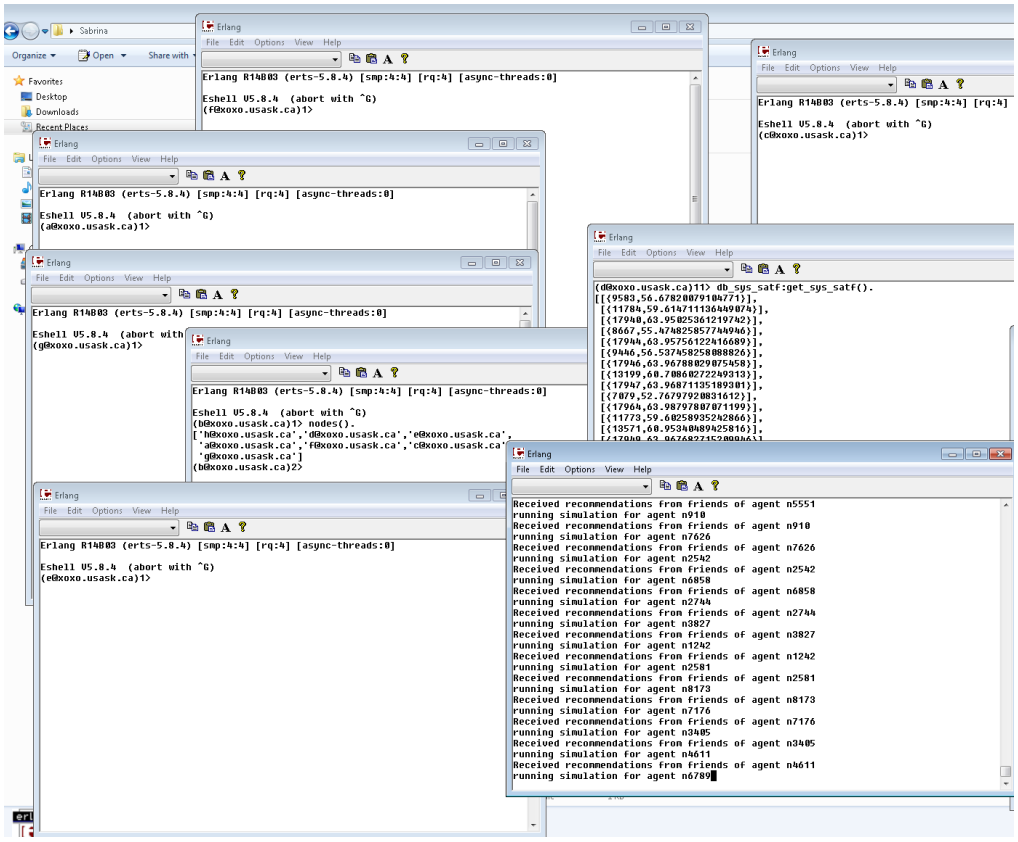

Figure 4-2: Snapshot of simulation with eight distributed Erlang systems

Query processing: Each agent, after being created, waits to send messages or to process information (see Appendix). An agent picked randomly, queries its friends about doctors. Friends respond to the query and also forward the message to their friends. The query propagates to the friends of friends only, because the trust in a message decreases with the number of hops it propagates.

The querying agent waits for a fixed period of time to receive messages. It doesn't process any information after that period. This is done in order to control the flow of information and to divide the time slots equally among all the agents. There may be some nodes in the graph with high connectivity (for example, more than 1000 connections) and an agent may have to wait for an indefinite time to receive recommendations after sending a query. Therefore, a time limit of 3 seconds is set in my simulation for each querying agent.

Processing doctor's information: After receiving the recommendation about a doctor from its friends, an agent creates its model of the doctor according to its trust & preference models by equation (1). If it is the first reference about this doctor then this data is stored in database of the agent. If there are existing models about the same doctor, then the old models are summed with the new model, average of all these is calculated and stored in the database.

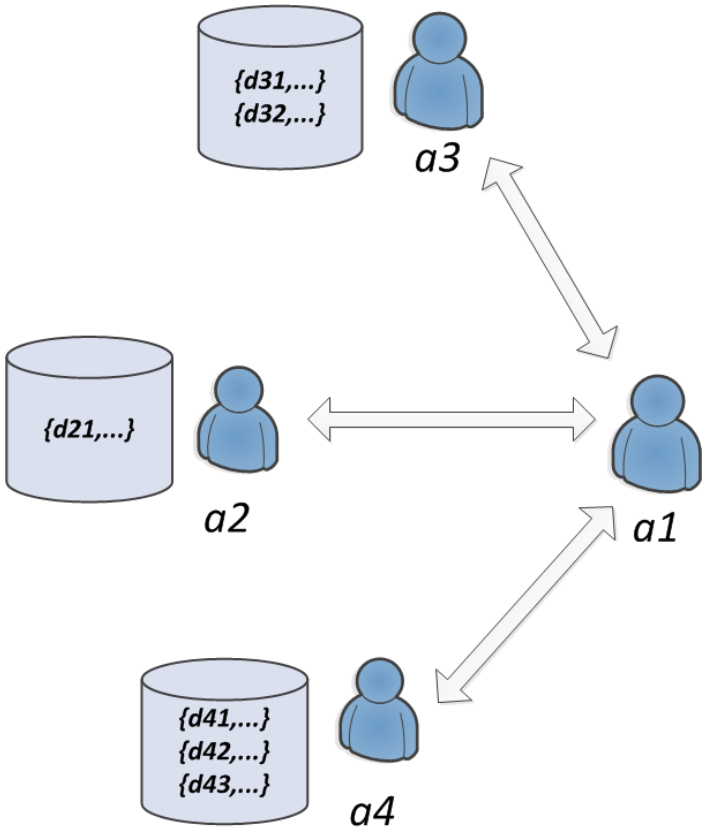

Figure 4-3: Agents with doctor's information in their own databases

Fig. 4-3 shows a scenario where agent *a1* is connected with three friends. Among its friends, agent *a3* has information about two doctors, *a2* has information about one doctor and *a4* has information about three other doctors.

Agent a1 asks its friends about their models of doctors and receive information about doctor d21, d31, d32, d41, d42 and d43. After getting all these recommendations, agent a1 checks which doctor has the highest trust value (calculated from equation (1) in Chapter 3) and also who

provided this information that is, the referee; and stores this information in its own database. Suppose, the trust value of doctor d42 was the highest among all the references agent a1 received. Then doctor d42's information will be saved in a1's database. This is shown in Fig. 4-4.

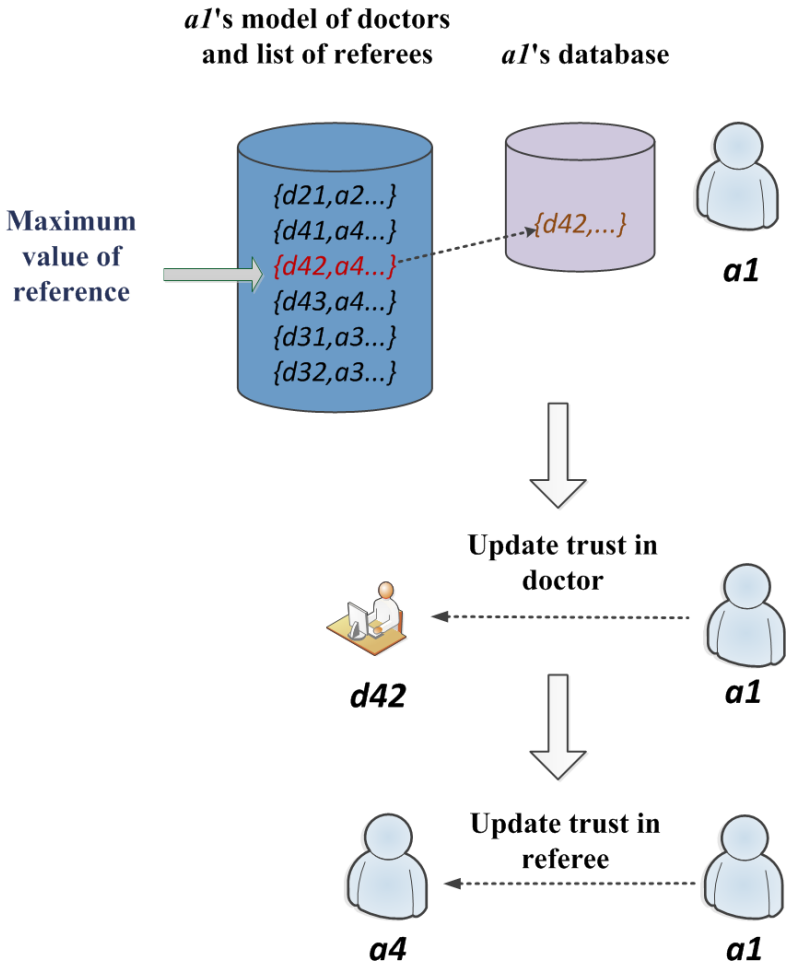

Figure 4-4: Storing doctor's recommendation and updating trust in the referee

Choosing the doctor to interact with: After all the above steps are done, the agent has information about at least one doctor in its database. However, it may have other doctors' information as well from previous interactions. Therefore, it sorts all the models of doctors based on the trust in them and picks one randomly from the top two to interact with. This is done in order to avoid picking the top-trusted doctor all the time. Consider the case where an agent has

information about two doctors in its database: one of them is a doctor it directly interacted with and another one it got recommendation about from its friends. Trust in the doctor the agent interacted directly with will usually have higher value than in the doctor it only got recommendation about. But the recommended doctor may be better suited for the agent. The second one from the top-trusted doctors may be better for the agent even in the case where all the models of doctors are based on recommendations only. As the trust value decreases with number of hops a message propagates, the recommendation (or, trust) value of a better suited doctor may be less than that of a doctor who is not very suitable but closer to the agent on the social network (because his recommendation propagated less number of hops). Therefore, by allowing it to choose one of the two top-trusted doctors, the agent is given more flexibility and the possibility of exploring and learning.

Updating trust in the doctor: Before interacting with the chosen doctor, the agent had some trust in the doctor based on either 1) recommendation from its friends or 2) its previous direct interaction with the doctor. In any case, the agent will have a new value of trust after the interaction. For the first case, the agent will discard the previous recommendation value in the doctor and store the new value of trust after its own direct interaction. For case two, where the agent had doctor's trust value in the doctor based on direct interaction before interacting this time, it will combine the previous trust with the trust value generated recently and calculate a new trust value with learning formula (4):

$t_{new}=t_{old}*\alpha + (1- \alpha)*evidence$……(4)

Here, α is the learning rate of the agent and evidence is interaction feedback skewed towards doctor's skill level. For example, if doctor's skill level on some criteria is 0.6 (in a scale between

0 and 1), after interacting with the doctor, *evidence* will be a random number between 0.4 and 0.8 (see Appendix).

Updating trust in referee: After interacting with the doctor and updating trust in him, the agent updates its trust in the referees who gave recommendations about this doctor (Fig. 4-4). If the agent's new model of doctor matches closely with that of referee, then agent's trust in referee will increase, otherwise, it decreases. This new trust value is stored for future reference (see Appendix).

Calculate the satisfaction of the agent: If *R* is a differentiated model to represent trust in the doctor after interaction and *P* represents the agent's preferences, then score (*A*) of the agent for current choice of doctor is calculated as follows:

$A_{current}=\sum_i R_i * P_i$, *where* $i \epsilon \{1, 2, 3\}$

After that, the satisfaction of the agent is calculated by equation (3). This is the main metric used in the evaluation of the proposed recommendation approach. As more requests are generated in the simulation, the system finds out how close the agents get to their best matches (computed in the first step of the simulation).

All the above steps are done after each request in the system. Following that, system's average satisfaction is calculated. The hypothesis that needs to be tested is that system satisfaction increases with the number of requests being processed (i.e. agents interacting with each other and exchanging references about doctors). To test the validity of this hypothesis, I experimented on two kinds of networks: a synthetic graph with small-world properties and a large scale real social network. Properties of the graphs, simulation setup, results and analysis of these experiments are described in the following chapter.

CHAPTER 5
EXPERIMENTS AND ANALYSIS

5.1 Simulation Hypothesis

The purpose of the evaluation is to check whether the agents will increase their satisfaction as a result of the interactions, and how close the matches found with this approach will fit the users' preferences ultimately. A good approach would allow the agents to learn quickly, so it is expected that the satisfaction would increase quickly and converge to a particular value that depends on the availability of doctors with features matching the preferences of the agents and the topology of the network (as the topology does not change during the experiment). A signal of convergence is that the increase in satisfaction gets smaller with the number of queries, and ultimately comes close to 0, i.e. the agents in the network have learned who the best doctors available in the network for them are.

5.2 Case Study 1: Experiment with Network of Kleinberg Model

5.2.1 Network Data

The first experiment is done on a synthetic network that exhibits the "small-world properties". Small-world networks have distinctive properties [40]:

- Most nodes are not neighbors of one another, but most pair of nodes can be reached by at least one short path
- There is an over-abundance of "hubs"-nodes that have a high number of connections
- Another property of small world network is that it has a clustering coefficient considerably higher than expected by random chance. The clustering coefficient is a measure of degree to which nodes in a graph tend to cluster together.

I experimented with a network generated by "KleinbergSmallWorldGenerator" [40] that has small world properties. The network I worked on had 49 nodes and 490 edges. The density of the network was 0.45, with a min degree of 9 and max degree of 14.

5.2.2 Simulation Setup

These experiments use an Intel Core i5 machine with a 3.20GHz processor and 4GB RAM.

The graph that was used for experiments had 49 nodes representing 49 agents. At the beginning of the simulation some of the agents knew few doctors. These experiments involved a population with a fairly high percentage of service providers (49 agents and 4 doctors); although later I experimented with lower percentage of service providers that match real population statistics. For this small graph, at the beginning, all the doctors were connected with equal number of agents (each doctor was connected with five agents).

There were preset initial trust values between any two agents, also preset trust values in each doctor from its patient and preset preference weights of all the agents. These values, between 0 and 1, were created with Erlang random number generator by calling: random:uniform(10)/10. In order to make the results reproducible, I had given certain seed values [49] to the random number generator so that it produces the same sequence of numbers across all runs and platforms. The simulation is run five times. Each of these 5 experiments is done on the same graph of 49 agents, but the initial trust values between any two agents or an agent and a doctor were different each time (depending on the seed value of random number generator). Also, the sequence of agents asking for doctors is dependent on the seed value for each experiment.

The seed values I used for five consecutive runs were as follows:

random:seed(1330,723213,346000).

random:seed(1330,729535,871000).

random:seed(5991,29821,991).

random:seed(1330,732690,759630).

random:seed(1330,18745,265000).

5.2.3 Results and Analysis

Fig. 5-1 shows the evolution of the overall level of satisfaction for all agents in the system with the number of requests. After a certain total number of requests in the system (for 49 agents, after about 250 requests), the system converges, meaning the agents find the best (or close to the best) doctors for themselves. The number of requests is plotted on the X axis and the percentage level of system satisfaction is plotted on the Y axis. The system satisfaction is around 20% in the beginning of the simulation and it grows quickly up to just above 80% for the first 120 requests and then it converges slowly to 90% after 250 requests.

As the system progresses and agents start learning, the standard deviation increases in the middle of the process and in the end it decreases indicating the convergence of satisfaction values. Means and standard deviations of 5 experiments are reported for each data point. It is clear from the plot that the percentage of system satisfaction converges (to almost 90%) with the number of requests.

To explain the convergent behavior, one has to consider that users choose to interact with a doctor with whom their preferences match and later update their trust in referees depending on their evaluation of the interaction. If the user is satisfied with the interaction, then trust in the referee increases, otherwise it decreases. These trust values determine whether any further references from these referees will have any impact on its future decision making. With the growing number of requests and therefore, more interactions between the agents, users are able to find trustworthy friends with whom their preferences match and also suitable service providers for them. The growing curve for system satisfaction in Fig. 5-1 implies that agents are able to

find the best (or close to the best) matches for themselves using my trust-based social network without the help of any central server.

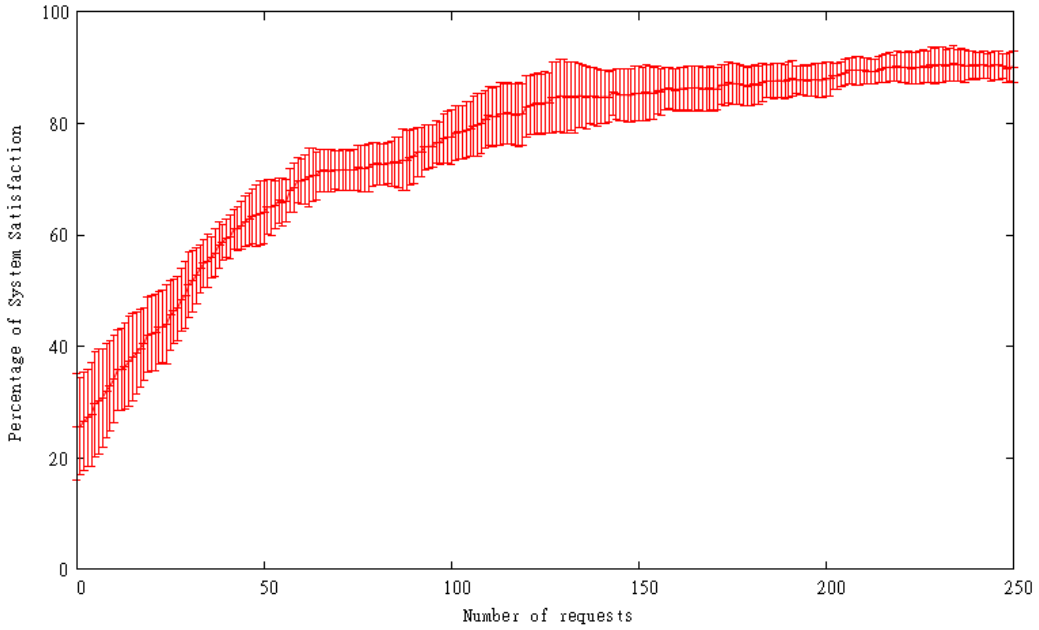

Figure 5-1: Percentage of system satisfaction for case study 1

5.3 Case Study 2: Experiment with Wikipedia Vote Network

5.3.1 Network Data

Social networks involve people and the relationship (like, friendship, transactions, communications etc.) between them [40]. Many social networks exhibit small-world properties described in section 5.2.1. I wanted to evaluate my approach by running simulation on a large-scale real social network. The social network graph used in the simulation is the "Wikipedia Vote Network", a large scale social network data available from Stanford Network Analysis Platform (SNAP) [42]. Nodes in the network signify Wikipedia users and a directed edge from node i to node j represents that user i voted on user j. There were 7115 nodes and 103689 edges in the graph. The average clustering coefficient is 0.2089. I wrote a java program to parse the

data file in order to provide input to the system (see Appendix). The network topology is static during the simulation, i.e. agents do not make new friends or lose friends.

5.3.2 Simulation Setup

The experiments use an Intel(R) Xeon(R) machine with two 2.33GHz processors and 16GB RAM.

At first, the input graph of 7115 nodes (representing 7115 agents in my model) is fed into the simulation. These experiments involve a population with eight service providers (doctors). This is done to match real world data, which shows there are 1.03 general/family physicians per 1000 people according to October 2011 data from Statistics Canada [41].

Similar to the previous experiment, there were preset initial trust values between any two agents, trust values in a doctor from its patients and preference weights of all the agents. These values, between 0 and 1, were created in Erlang random number generator by calling: random:uniform(10)/10. The seed value I used for random number generator was:

random:seed(1330,732690,759630).

5.3.3 Distribution of Patients

As it wasn't possible to find any statistical data about the distribution of patients over doctors, I used normal distribution since this is one of the most common found distributions among statistical and natural phenomena [50]. Three experiments I present in this section involved the normal distribution of patients for each of the eight doctors with a mean number of connections of 350 patients, 500 patients and 750 patients respectively.

5.3.4 Results

The goal of the experiment was to see whether using my approach of recommending services in a trust-based network helps agents to achieve better satisfaction. I experimented with different number of doctor-patient connections. The mean number of connections from a doctor to patient

in the initial configuration varied from 350 to 750. The results are plotted in Fig. 5-2. The number of requests generated in the system is plotted in horizontal (X) axis and percentage of system satisfaction (which is the average satisfaction of all the agents of the system) is plotted in the vertical (Y) axis. The result of the experiment with initial mean connection of 350 is drawn with a red line and "+" symbol in every 2000th point. The green line with "•" symbol in every 2000th point indicates the result of experiment with initial mean connection of 500 and the blue line with "▲" symbol shows the result of experiment with initial mean connection of 750. From Fig. 5-2, the evolution of the overall level of satisfaction can be observed for all agents in the system with the number of requests for all three experiments.

5.3.5 Analysis

The purpose of the evaluation was to check how close the matches found with this approach fit users' preferences and increase their satisfaction. From Fig. 5-2 I can see that, the system satisfaction increases very quickly with the number of requests until the satisfaction converges to a final value. For 500 connections, the system satisfaction is around 32% in the beginning of the simulation and it grows quickly until 35000 requests, then it converges slowly to around 75%. For mean connections of 350 and 750, the initial satisfaction values are 24% and 43% respectively, and both these curves converge to the satisfaction value of around 74%. Therefore, even with different initial configurations, all these experiments reach a convergence value of around 75%.

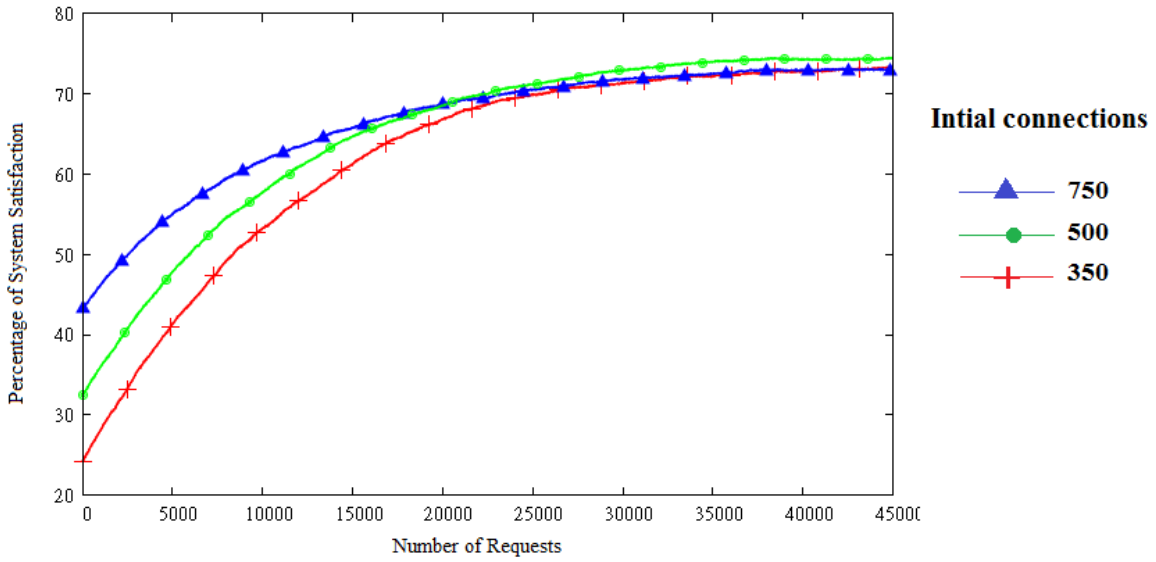

Figure 5-2: Percentage of system satisfaction for case study 2

As more requests are generated in the system, some relations become more trusted, some less-trusted. If the user is satisfied with the service, the trust value in referee also increases, otherwise it decreases. It implies that trust in friends who have different preference criteria will decrease with time and will have less effect in decision making. As agents update their trust models with each request, they learn to find better matches according to their preferences, and their level of connectedness. The system satisfaction will not reach 100%, as the simulation uses a very large and realistic social network. An agent "on one side of the network" may not get information about a well matching doctor who is connected with the agents "on the other side" of the network. As the system is checking how close an agent gets to the best possible doctor in a large network and as the trustworthiness of a message decreases with the number of hops propagated, the recommendation value of that doctor might be too low for the agent to actually choose to interact with the doctor.

Another interesting point to note is, even after starting with a lower initial satisfaction value (32%), the final satisfaction value for the experiment with 500 connections (74.5%) is slightly

higher than that of the experiment with 750 connections (73.3%). My speculation for this is based on two important features of the model: an agent considers one of the two top-trusted doctors to interact with, and trust in a doctor it interacted directly before is usually higher than any recommendation value. Suppose, for agent "*x*", doctor *d1* is better than doctor *d2*. Consider two different experiments. For the first experiment, in request number *n-1*, the agent has only *d2*'s information in its database and in request number *n* it gets recommendation about *d1*. It may choose to interact with *d2* again, not taking the recommendation into consideration. Therefore, the satisfaction of agent *x* increases slightly after run number *n*, resulting in very small rate of increase in the overall satisfaction. On the other hand, in experiment two, the agent may not have any doctor's information in its database in request number *n-1*. The system satisfaction of this experiment in this request number is lower than that of experiment one, where agent *x* had some satisfaction value, contributing in the overall system satisfaction. However, after the agent gets recommendation about doctor *d1* in run *n*, it chooses to interact with this doctor and gets higher satisfaction from him than it could get from doctor *d2*. It improves the overall system satisfaction as well. Therefore, even after starting with a lower system satisfaction, the rate of increase can be higher in the experiment depending on the information an agent has and the decision it makes. The growing curve for system satisfaction implies that agents are able to find quite good matches for themselves in a trust-based social network without the help of any central server.

5.4 Evaluating the Robustness With Respect To Malicious Referees

In a real network there can be malicious agents who try to mislead other agents by providing unfair ratings. A good trust and reputation approach (or a level 2 approach, according to Sierra and Sabater's classification [36]) should be able to allow agents to quickly identify these undesirable agents and avoid them, choosing more trustworthy referees. In order to check the robustness of my approach against malicious agents, a simulation of my approach is carried out,

introducing some malicious agents in the system. The social graph I used for this was the "Wikipedia Vote Network" [42] mentioned in section 5.3.1.

5.4.1 Main Idea

The main idea of this experiment is to test whether the reputation of malicious agents decreases with the number of interactions in the system. "Reputation" is defined as the average trust that all contacts of a particular agent have in it. A certain number of malicious agents are introduced randomly in the network, who always give top trust values for a particular doctor, whose capabilities are low. In this way a collusion aiming to promote a particular bad service provider is simulated. The hypothesis is that in the course of the simulation, as new requests are generated, the reputation of the malicious agents will decrease whereas the average reputation of other "referees" will stay the same or increase, as they provide honest recommendations.

5.4.2 Experimental Setting and Measure

To simplify the calculation of reputation and the analysis of the experimental results, I sum up the three trust values of a differentiated trust model, thus creating a general (non-differentiated) trust value $TG_{ab}=TP_1 + TP_2 + TP_3$, (using the notation introduced in Figure 3-2), which can take values in the interval [0.3, 3]. This is because in my model, the highest level of trust that an agent can have in another as a referee is 1 and the lowest level of trust can be 0.1 in each of the three criteria, so the sum of the three values can vary between 0.3 and 3. Therefore, the initial reputation of the agents must be a number between 0.3 and 3.

The simulation compares the average reputation of malicious agents with other agents who provide honest recommendations. There were 15 malicious agents introduced in the system. These agents were picked by the random number generator with seed:

random:seed(1331,574695,772000).

These malicious agents provide dishonest recommendations about one particular doctor (suppose, "*d1*") of the system, giving very high ratings in all the criteria, although this doctor's actual skill levels are very low. As the friends of these malicious agents query them about doctors, these malicious agents provide recommendations, but with falsified information. Also, it is assumed that, initially, only these malicious agents know about doctor *d1*. So all the recommendations received about *d1* at first will be falsified. If in the course of the simulation any of the other agents gets to interact with *d1*, it will develop its own model of *d1*, and it will share these values when asked for recommendations by its friends. I want to check how quickly the friends (neighbor nodes) of the malicious agents will identify these undesirable agents and decrease their trust in them, i.e. how quickly the reputation of the malicious agents will drop.

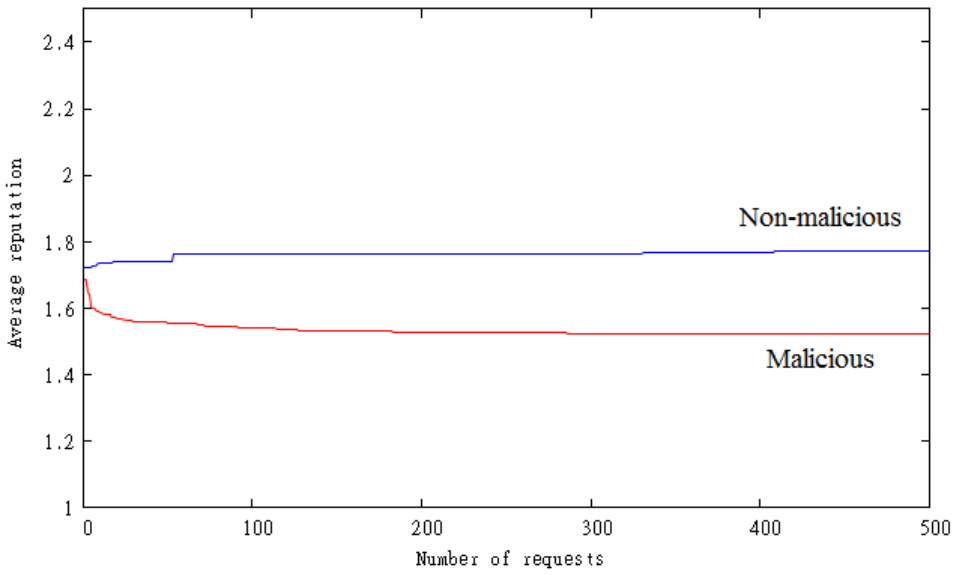

Figure 5-3: Comparison between the reputation of good and bad agents

For comparison, the same simulation is done without malicious agents, i.e. all agents are truthful in giving recommendations. The same agents who are friends of these 15 agents (but in this case these 15 agents are truthful) query about doctors and receive recommendations. Fig. 5-3

shows the results of both simulations for 500 queries. The average reputation of good agents is drawn with a blue line and the average reputation of bad agents is drawn with a red line.

5.4.3 Analysis

As it can be seen from Fig. 5-3, the average reputations of good agents and malicious agents start with almost the same value. According to my trust model, after receiving recommendation from a referee, an agent increases its trust in the referee if it finds the recommendation useful; otherwise it decreases its trust in the referee. This updated trust value is stored in the agent's database and it determines how future references from this referee agent will be evaluated. If the agents are satisfied with the recommendations received from these referee agents, their reputation will increase or, at least, continue to be high.

As it could be observed from Fig. 5-3, within the first few interactions in the system, the average reputation of the 15 malicious agents decreases, and then it stays steady at a lower value. If any of the agents in the system considered the references from one of these agents, it would increase or decrease its trust in the referee, thus changing its reputation. However, as it could be observed from Fig. 5-3, the average reputation of the 15 malicious agents only decreases with the number of requests and then becomes stable. The stabilization of the reputation score implies the recommendations from the malicious are not considered after a certain number of requests. Their friends learn about the low trustworthiness of these malicious referees and do not select the recommended doctor from them. Therefore, the trust values in these malicious agents do not change. Therefore, the bad agents cannot have much influence on their friends in decision making after first approximately 25 interactions. Consequently, the system is able to work with more trustworthy users, nullifying the effect of the malicious agents.

5.5 Summary

The aim of the experiments was to test the validity and performance of my trust network. The evaluation measures were defined accordingly to examine how users of the system find better matches for themselves. In order to evaluate the performance of my model, I chose two different types of social graphs and analyzed the effect on both small and large networks.

From the simulation results, it could be observed that the average satisfaction level of the agents in the system increases significantly for both the graphs in Fig. 5-1 and Fig. 5-2. Fig. 5-3 shows the robustness of my approach, how it deals with malicious agent by decreasing their reputation with the number of requests in the system. The experiments were conducted with different seed values to nullify any kind of bias that could affect the outcomes. The results reveal that, regardless of initial configurations, by using the proposed differentiated trust approach, agents can learn about the trustworthiness of friends and find more suitable service providers for themselves according to their preferences. The growing curves in Fig. 5-1 and Fig. 5-2 indicate that starting with any existing real social network; agents are being able to find better matches for them by reinforcing or weakening their trust-relationships with their connections, depending on their preferences and usefulness of the recommendations received.

CHAPTER 6
CONCLUSION

6.1 Summary

The purpose of this research is to employ trust and reputation mechanisms in decentralized user modeling system to recommend service providers to the users. I presented a preference-based recommender system, using trust mechanisms. I evaluated the performance of my approach by conducting simulation in both small and large social networks. From the results of the simulation, it can be observed that my trust-based recommender system supports the hypothesis presented in section 5.1 and is able to find better matches for its users according to their needs and preferences.

My approach enriches two different research areas: trust and reputation mechanisms and decentralized user modeling. This approach has an evolutionary aspect: agents representing users in a decentralized environment will learn from user's feedback to improve their user models and the trust models in other agents representing users. Thus, starting with an existing real social network, certain relationships will become more trusted and some - less trusted, depending on the preferences of the users and the usefulness of recommendations exchanged. I am optimistic that this system, when implemented in a real expert finding system, will prove to be efficient, effective and psychologically acceptable to users, as it exploits trusted social networks of friends to generate recommendations.

6.2 Contributions

This thesis makes several contributions to the areas of user modeling and trust and reputation mechanisms:

- User Modeling:
    - A practical approach for decentralized user modeling; agents storing user's personal information without disclosing them into any central server.
    - A novel approach for user modeling: allowing agents to share information and selectively interpret the received information, to update user models and to make decisions based on trust mechanisms.
- Trust and Reputation mechanism
    - Differentiating user's trust in the referees
- The development of a new Erlang-based simulation system, capable of using large social graphs as input and distributing the computation across different machines.
- Evaluation of the approach involving a realistic data-set (large social network). This kind of evaluation is not typically done in user modeling, but is found in trust and reputation mechanism evaluation and in some recommender algorithms evaluation.

Although I used doctor-patient scenario as an example, this approach can be applied in any expert finding or recommender system. For example, in a network of users looking for books according to their tastes and preferences, my model can be applied to recommend to users books according to their preferences and trust in the recommendations provided by their friends. I believe that this approach is equally applicable in such scenarios where users form a social network based on their common interest and search for suitable experts or resources.

6.3 Possible Implications

Pure reputation based approaches lead to network effects, i.e. a few most reputed service providers attracting most of the clients. With a pure (centralized) reputation based approach, most patients will go to few doctors who have high trust values (leading to long waiting times) whereas many doctors will have very few patients (long tail distribution). My approach, on the other hand, may increase diversity by spreading the distribution of patients across the whole set of doctors based on patients' needs.

Consider the doctor-patient scenario discussed in chapter 3. Patients may be mothers of young children who are interested to find good specialist doctors (pediatricians). There are numerous social websites that allow patients to form networks amongst them depending on common interest, disease etc. (e.g. PatientsLikeMe [30]). My approach can be implemented in such domain-specific networks. In these networks, naturally, some doctors have more competence/higher expertise levels than others. Some are close by, while others are far away, some are friendly, and some are less so. The patients may care also about the doctor's gender or nationality (e.g. immigrants who can't speak well the language of their new country may prefer to go to a doctor who speaks their native language). In traditional trust-based networks where trust has one single value, everyone will consider the most competent doctors to be more trustworthy. Patients will try to go to the few doctors with the highest trust value (based on perceived competence resulting either from years of experience, or credentials), which will lead to "network effects", i.e. the emergence of hubs, resulting in longer waiting times and less availability of these more competent doctors for really serious cases. However, often the condition of patients is not severe and other doctors are well-suited to handle these patients in a timely manner. In this case patients will prefer to see a doctor who is available at that point of time rather than waiting for the top trusted doctor. My approach, because it is based on a multi-

faceted trust model, allows patients to set different weights on different preferences, e.g. the patient's preference weight could be high (closer to 1) in the criterion "doctor's availability". Similarly, all other patients can specify their preference weights and find doctors who are suited according to their needs. This can lead to shorter waiting times and more diversity in doctor-patient connections. As our approach is applicable in any service recommendation scenario, it can bring the above described benefit of regulating supply and demand and alleviating undesired network effects of concentration. In principle, any personalized service selection mechanism would have this effect; however, most service recommender systems require a central point where all preferences of clients are collected. This leads to privacy issues. My approach avoids these issues by not requiring clients to reveal their own preferences.

In the medical domain, doctors are of two types: family physicians and specialists. In most cases, if the condition of patients is not that severe, they go to the family physicians. However, in critical situations people need to see specialists. In such cases, family physicians can serve as referees. They will know more specialists than any common person and can recommend experts or specialists to the patients. They work as "hubs" that have a good number of connections with patients and contribute to the system with references of other doctors. And after the patients get treatment from the specialists, they can give feedback to the physicians about their experience they had with the specialists. The physicians can then work as local reputation servers by aggregating all the information and building reputation of the specialists. These local reputation servers can provide better references later to the users. This is similar to the super-peer trust and reputation management approach proposed by Wang & Vassileva [50], and future work will explore in more details how differentiated trust in referees can be integrated in a super-agent - based trust and reputation management.

6.4 Limitations and Future Work

I tried to collect statistical information in order to feed our simulation with realistic data. Information I wanted to collect included: on average, how many different patients a physician sees over a period of time, for example, a month or a year in a specific city or town, what kind of network patients have among themselves etc. I contacted Ms. Shawna Weeks, research approval coordinator of Saskatoon Health Region, Dr. Laurence Givelichian, Head of dept. of Pediatrics, Royal University Hospital , Dr. Colum Smith, Vice-President, Clinical Services and Senior Medical Officer, Saskatchewan Cancer Agency, Dr. Vernon Hoeppner, Head of dept. of Medicine, Saskatoon Health Region, Dr. Alan Casson, Professor and Head of Surgery, Saskatoon Health Region, Dr. Krista Baerg, Consultant Pediatrician in the division of General Pediatrics, Royal University Hospital, Dr. J. Akhtar, Clinical Associate Professor of Medicine in General Cardiology and Echocardiography, University of Saskatchewan and several other doctors and websites by email or phone. It took me several weeks to communicate with all the contact persons and their referrals. Most of them were willing to contribute in my work. However, the data I was looking for was almost impossible to compile and in the end, I could not get any statistical information, except some anecdotal data (estimates) received from health professionals.

Due to the lack of any real statistics, I worked with several possible values based on anecdotal data. I tried with different number of links and connectivity. The number of connections for a doctor varied from 350 to 750 over the experiments because of anecdotal data (rough estimates) mentioned by several people I and my supervisor talked to: 500 connections per doctor. The distribution of patients (agents) over the set of the doctors (service providers) followed normal distribution. This was done because many variables in real life (e.g. test scores, height etc.) follow normal distributions [51]. I experimented with different seed values and different initial

configurations of the graphs so that bias in the experiments could be avoided. The results show that regardless of connectivity and initial configuration of graphs, the agents are capable of learning using their trust-network and find better matches for them. I believe that, if this simulation is conducted with real-life data, it will continue to help users find experts without the help of any central server and without divulging their private information, like, preferences and trust values. Another limitation that I had to impose in the experiments was the static nature of the social graphs used. For future work, the model can be implemented in a scenario where the network topology is not static; agents can make new friends or lose existing connections.

In future, if this model is implemented in a real-world application, then each user will have a client application (agent) running for her either on her device or on a server/cloud. The user should be able to connect to her friends. So they also should run their own agents. She will have to provide the agent with her preferences and trust in her friends. The agent, on behalf of the user, should communicate with other agents to know about their models of service providers. The user will then select one service provider based on the references her agent can get for her. After interaction with the service provider, she will inform the agent about her evaluation. The agent will then update the trust relations with her friends based on the user's feedback. This kind of applications can be useful for e-health, online consultations and expert finding systems where users, connected in a decentralized environment, are looking for personalized services.

The idea of sharing information among users along trusted relationships in their social networks and filtering this information according to the users' preference and trust models is novel. I hope that, if this system is built as a real-world application, it will prove to be an effective trust-based recommender system to help users find suitable service providers in a decentralized environment.

LIST OF REFERENCES

1. Abdul-Rahman, A. and Hailes, S.: Supporting trust in virtual communities. *Proceedings of the Hawaii International Conference on System Sciences* (2000).

2. Adomavicius, G. and Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17 (6), 734-749 (2005).

3. Canadian BBB, http://www.bbb.org/canada/, last accessed 22nd March, 2012.

4. Carter, J., Bitting, E., and Ghorbani, A.: Reputation Formalization for an Information-Sharing Multi-Agent System. *Computational Intelligence 18* (2), 515-534 (2002).

5. Chatfield, C., Carmichael, D., and Hexel, R.: Personalisation in intelligent environments: managing the information flow. *Proceedings of the 19th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: citizens online: considerations for today and the future. ACM International Conference Proceeding Series*, vol. 122, pp. 1–10 (2005).

6. Crumlish, C. and Malone, C.: *Designing Social Interfaces*, O'Reilly Press (2010).

7. Dell'Amico, M. and Capra, L.: Sofia: Social filtering for robust recommendations. *Trust Management II*, 135–150 (2008).

8. Erlang Programming Language, http://www.erlang.org/, last accessed 13th April, 2012.

9. Erlang – dets, http://www.erlang.org/doc/man/dets.html, last accessed 18th March, 2012.

10. Erlang – gen_server, http://www.erlang.org/doc/man/gen_server.html, last accessed 18th March, 2012.

11. Erlang—random, http://www.erlang.org/doc/man/random.html#seed-3, last accessed 20th March, 2012.

12. Fink, J. and Kobsa, A.: A review and analysis of commercial user modeling servers for personalization on the world wide web. *User Modeling and User-Adapted Interaction* 10 (2), 209–249 (2000).

13. Hang, C.W., Wang, Y., and Singh, M.P.: Operators for propagating trust and their evaluation in social networks. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems*, 1025–1032 (2009).

14. Heckmann, D. and Krueger, A.: A user modeling markup language (UserML) for ubiquitous computing. *Proceedings of the 8th International Conference on User Modeling (UM'2003)*, Springer, 393-397, (2003).

15. Heckmann, D., Schwartz, T., Brandherm, B. and Kröner, A.: Decentralized user modeling with UserML and GUMO. *Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM), 9th Intl Conference on User Modeling, Edinburgh, Scotland*, 61-64 (2005).

16. Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K.: The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. *Proceedings of the fourteenth Conference on Uncertainty in Artificial Intelligence*, Citeseer , 256–265 (1998).

17. Josang, a, Ismail, R., and Boyd, C. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43, 2, 618-644 (2007).

18. Kamvar, S. and Schlosser, M. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th International World Wide Web Conference*, 640-651(2003).

19. Kay, J. Stereotypes, student models and scrutability. *Intelligent Tutoring Systems*, 19-30. (2000).

20. Kay, J., Kummerfeld, B., and Lauder, P.: Personis: a server for user models. *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, 203-212 (2002).

21. Kobsa, A.: Generic User Modeling Systems. *User Modeling and User-Adapted Interaction*, 49-63 (2001).

22. Kobsa, A.: *User Modeling and User-Adapted Interaction* 1: v-viii, 1991. (~) 1991 Kluwer Academic Publishers. Printed in the Netherlands. Linguistics (1992).

23. Kobsa, A.: Privacy-enhanced web personalization. *The adaptive web*, Springer-Verlag, 628–670 (2007).

24. Kuter, U. and Golbeck, J. Sunny: A new algorithm for trust inference in social networks using probabilistic confidence models. *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1377-1382 (2007).

25. Marsh S. Formalising Trust as a Computational Concept, Ph.D. Thesis, University of Stirling, (1994).

26. Matsuo, Y. and Yamamoto, H.: Community gravity: measuring bidirectional effects by trust and rating on online social networks. *Proceedings of the 18th international conference on World wide web*, ACM, 751–760 (2009).

27. Mui, L., Mohtashemi, M., and Halberstadt, A.: A computational model of trust and reputation. System Sciences, 2002. HICSS. *Proceedings of the 35th Annual Hawaii International Conference on, IEEE*, 2431–2439 (2002).

28. Niu, X., McCalla, G., and Vassileva, J.: Purpose-Based Expert Finding in a Portfolio Management System. *Computational Intelligence*, Vol. 20, No. 4, 548-561 (2004).

29. Nusrat, S. and Vassileva, J.:Recommending Services in a Trust-Based Decentralized User Modeling System. *Advances in User Modeling.* UMAP 2011Workshops, LNCS, Vol. 7138, Springer, 230-242, (2012).

30. PatientsLikeMe, http://www.patientslikeme.com/ last accessed 20th March, 2012, last modified 20th March, 2012.

31. Qiu, X., Zhang, L., Wang, S., and Qian, G.: A Trust Transitivity Model Based-on Dempster-Shafer Theory. *Journal of Networks*, 5(9), 1025-1032 (2010).

32. Rasmussen, L. and Jansson, S.: Simulated social control for secure internet commerce. *Proceedings of the New Security Paradigms Workshop*, ACM, 18-25 (1996).

33. Resnick, P. and Zeckhauser, R.: Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. *Advances in Applied Microeconomics: The Economics of the Internet and E-Commerce*, vol. 11, In M. Baye, Ed., 127–157, (2000).

34. Rich, E.: User modeling via stereotypes. *Cognitive science* 3, 4, 329–354 (1979).

35. Rich, E.: Users are individuals: individualizing user models. *International journal of man-machine studies*, 199-214 (1983).

36. Sabater, J. and Sierra, C. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review* 24, 1, 33-60 (2005).

37. Sensoy, M., Zhang, J., Yolum, P., and Cohen, R.: Poyraz: Context-aware service selection under deception. *Computational Intelligence* 25, 4, 335–366 (2009).

38. Shoham, Y. and Balabanovic, M.: Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM* 40, 3, 66-72 (1997).

39. Sleeman, D.: UMFE: a user modelling front-end subsystem. *International Journal of Man-Machine Studies* 23, 1, 71-88 (1985).

40. Social Network Generation, http://www.infovis-wiki.net/index.php/Social_Network_Generation/, last accessed 20th March, 2012.

41. Statistics Canada, http://www.statcan.gc.ca/, last accessed 27th March, 2012, last modified 20th March, 2012.

42. Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/, last accessed 20th March, 2012.

43. Vassileva, J.: A classification and synthesis of student modelling techniques in intelligent computer-assisted instruction. *Proc. of 3rd International Conference, ICCAL'90*, Berlin, Springer-Verlag, 202–213 (1990).

44. Vassileva, J.I., Greer, J.E., McCalla, G.I.: Openness and Disclosure in Multi-agent Learner Models. *Proceedings of Workshop on Open, Interactive, and Other Overt Approaches to Learner Modelling, International Conference on AIED*, Le Mans, France, 43-49 (1999).

45. Villano, M.: Probabilistic student models: Bayesian belief networks and knowledge space theory. *Intelligent Tutoring Systems*, Springer, 491–498 (1992).

46. Wang, Y. and Vassileva, J.: Trust and reputation model in peer-to-peer networks. *Proceedings of Third International Conference on Peer-to-Peer Computing (P2P2003)*, 150-157 (2003).

47. Wang, Y. and Vassileva, J.: Bayesian network-based trust model. *IEEE/WIC International Conference on Web Intelligence*, IEEE, 372–378 (2003).

48. Wang, Y. and Vassileva, J.: Trust-based community formation in peer-to-peer file sharing networks. *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, IEEE Computer Society, 341–348 (2004).

49. Wang, Y. and Vassileva, J.: Toward trust and reputation based web service selection: A survey. *International Transactions on Systems Science and Applications 3*, 2, 118–132 (2007).

50. Wang, Y.: Trust and Reputation Management in Decentralized Systems, Ph.D. Thesis, University of Saskatchewan, (2010).

51. Wikibooks: Signals ad Systems/ Common Distribution, http://en.wikibooks.org/wiki/Signals_and_Systems/Common_Distributions#Gaussian_Distribution, last accessed 22nd March, 2012, last modified 12th April, 2011.

52. Yimam-Seid, D. and Kobsa, A.: A. Expert-Finding Systems for Organizations: Problem and Domain Analysis and the DEMOIR Approach. *Journal of Organizational Computing and Electronic Commerce* 13, 1, 1-24 (2003).

53. Yu, B. and Singh, M.P.: A social mechanism of reputation management in electronic communities. *Proceedings of the 4th International Workshop on Cooperative Information Agents*, 154-165 (2000).

54. Yu, B. and Singh, M.P.: An evidential model of distributed reputation management, *Proceedings of the First Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, ACM, 294-301 (2002).

APPENDIX
CODE SNIPPETS

1 Creation of Agents in Distributed Erlang System

If agent n30 has three other friends, n1412, n3352 and n5254 respectively, then the following command creates a process representing agent n30 which has three connections/friends:

```
create_nd(n30, [n1412,n3352,n5254]).

create_nd(Name, List) ->
        case lists:member(Name, [n0,n1,n30]) of
        true->
                PID=spawn('a@xoxo.usask.ca',query_nodes_d,node_loop_d,[Name,List]);
        false->
                % create rest of the agents in other Erlang runtime systems
        End.
```

Figure A1: Erlang code snippet for creation of agents

This block of code implies agent n0, n1 and n30 are created in Erlang node (executing Erlang runtime system) "a@xoxo.usask.ca". Other agents are created in different other Erlang nodes. These nodes are then connected by net_adm:ping(Node).

2 Query Processing

The following code snippet shows the loop where an agent waits to receive messages and forwards messages to intended recipients:

```
%each agent has a name and the list of friends it's connected to

node_loop_d(Name, List) ->

%receive either a message to forward query or to process information
receive

% forward query
{forward, S, Parent, GrandP, TTL, Visited, Sent_time} ->


                forward(S, Parent, GrandP, TTL, [Name]++lists:flatten([Visited]), List,
Sent_time),
```

```
            node_loop_d(Name, List);

%process information
{process_data, Referee, Agent, P, TTL, [QueryResult],Sent_time} ->


            L=tuple_to_list(QueryResult),
            %%% do the rest

            %% Get the results of query from your friends and process the information

            node_loop_d(Name, List)
end.

%there is a function forward/7 defined that handles forwarding
%the query to the intended recipients depending on the number of
%hops and time the message passed. It also handles the extreme
%situations in which the node is connected to no other node or
%only one node

forward(S2, Parent, GrandP, TTL, V, [H|Rest],Sent_time) ->

      case lists:member(H, V) of %% is this node visited already?

            false -> %%if not, check database

            Result_2= db_doc_list:lookup_for_agent(S2),
            call_for_process_d(S2, Parent, GrandP, 0, Result_2,Sent_time);

            global:whereis_name(H)!{forward,H,S2,Parent,(TTL-1),V,Sent_time},
%%forward again
         forward(S2,Parent,GrandP,TTL,[H]++lists:flatten([V]),Rest,Sent_time)

%%rest
      end.
```

Figure A2: Erlang code snippet for forwarding query

```
TRY=lists:flatten(db1:get_model_from_ref_of_doc_for_agent(Agent, Referee,
Doc)),
            case TRY of

            []->

            %io:format("~nStoring new recommendation for comparison... ~n"),

           db1:insert_model(Agent, Referee, Doc, R);

            OldModel->
%io:format("~nDifferent data for the same doctor, so adding with old data and
taking average... ~n"),
```

```
              Old_Ref=lists:flatten(OldModel),

              Old_Ref_cnt1 =
lists:flatten(db1:get_num_references_from_referee(Agent, Referee, Doc)),
             [Old_Ref_cnt]=Old_Ref_cnt1,

             Old_Ref_mult= maths_d:multiplyVec_withNum(Old_Ref, Old_Ref_cnt),

             Sum_ref_all= maths_d:vSum(Old_Ref_mult, lists:flatten(R)),

             New_Ref_cnt= Old_Ref_cnt+1,

             New_ref_value= maths_d:divListByNumber(Sum_ref_all, New_Ref_cnt),



      db1:insert_updated_delete_old(Agent, Referee, Doc, New_ref_value,
New_Ref_cnt)



end
```

Figure A3: Erlang code snippet for storing data after receiving recommendation

3 Mathematical Functions

```
% Multiplication of vectors (element-by-element) containing same number of
%elements
vMul([], [], []) ->
[];

vMul([H1 | T1], [H2 | T2], [H3 | T3]) ->
[H1 * H2 * H3] ++ vMul(T1, T2, T3).

% Summation of vectors containing same number of elements
vSum([],[])->[];
vSum([H1 | T1], [H2 | T2]) ->
[H1 + H2] ++ vSum(T1, T2).

% Multiplying vector elements with a number
multiplyVec_withNum([],_Num)->[];
multiplyVec_withNum([H1|T1],Num)->
      [H1*Num]++multiplyVec_withNum(T1,Num).

% Adding number to vector elements

addVec_withNum([],_Num)->[];
addVec_withNum([H1|T1],Num)->
      [H1+Num]++addVec_withNum(T1,Num).


%Find the maximum numbered element from a list of lists
```

```
max([]) ->
    [];
max([H|T]) ->
    max(H, T).

max(M, []) ->
    M;
max(M, [H|L]) when M > H ->
    max(M, L);
max( M, [H|L]) ->
    max(H,L).

%Find the standard deviation

std_dev(Values, Avg) ->
    Sums = lists:foldl(
        fun(V, Acc) -> D = V - Avg, Acc + (D * D) end,
        0, Values),
    math:sqrt(Sums / (length(Values) - 1)).
```

Figure A4: Erlang code for various numerical operations

```
%%generating random number skewed towards a given number

generateRandom([])->[];
generateRandom([H1|T1])->

      case ((H1-0.2)+ 0.4*random:uniform(10)/10) of
            Neg when Neg<0 -> F=0.001;
            GreaterThanOne when GreaterThanOne>1 -> F=0.999;
            _True->F=_True
      end,

      [F]++generateRandom(T1).
```

Figure A5: Erlang code for generating random numbers

```
%Agent's Model of doc, Referee's Model of doc, Agent's trust in referee,
Alpha
calculate_trust_friend([],[],[],_V)->[];
calculate_trust_friend([H1|T1],[H2|T2],[H3|T3], Alpha)->
      case H1-H2 of
            Pos when Pos > 0 -> Evidence=1-(H1-H2);
            _-> Evidence=1-(H2-H1)
      end,
      %io:format("~nPrinting Evidence : "),
      %io:write(Evidence),
      [H3*Alpha + (1 - Alpha)* Evidence]++calculate_trust_friend(T1,T2,T3,

Alpha)
```

Figure A6: Erlang code for updating trust in friends

```
calculate_percentage(A, D, N, Num_agent)->

      AP=db_pref:get_pref_of_agent(A),

      AT=db_doc_list:get_doc_trust_vector(A, D),

      Mul_v=maths_d:vMul(lists:flatten(AP), lists:flatten(AT), [1,1,1]),

      Present_Score_agent=maths_d:eSum(Mul_v),

      Max_score_agent1= lists:flatten(db_max_scores:get_max_score_of_agent(A)),

      [Max_score_agent]=Max_score_agent1,

      Div_score=(Present_Score_agent/Max_score_agent)*100,

      db_agent_satf:insert_data(A, Div_score),

      Elem_set=db_agent_satf:get_satf_of_agents(),
      Elem=lists:flatten(Elem_set),
      Sum_satisfaction=maths_d:eSum(Elem),

      Avg_satisfaction=Sum_satisfaction/Num_agent,

      db_sys_satf:insert_data(N, Avg_satisfaction)
      .
```

Figure A7: Erlang code snippet for calculating system satisfaction

4 Parsing Data File

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.*;

public class Parse_SN
{
      public int NUM_NODES=7115;

      public int LINKS=1500;

      public int[] countOfLinks=new int[NUM_NODES];

      public int[] numOfLinks=new int[NUM_NODES];

    public long[] elements=new long[NUM_NODES];

    public long[][] elements_links=new long[NUM_NODES][];

    Hashtable numbers = new Hashtable();
```

```
public int TOTAL_NODES=0;

public Parse_SN()
{
	try
	{
		FileReader input = new FileReader("new.txt");
		BufferedReader bufRead = new BufferedReader(input);
		String line;	// String that holds current file line
		String delims = "[,]";
		int count = 0, i; // Line number of count

		for(i=0; i<NUM_NODES;i++)
		{
			countOfLinks[i]=0;
			numOfLinks[i]=0;
		}

		String[] tokens;
		tokens=new String[2];

		String[] tokens2;
		tokens2=new String[2];
		// Read first line
		line = bufRead.readLine();


		for(i=0; line != null; i++)
		{
			tokens= line.split(delims);
			do
			{

			++count;

			line = bufRead.readLine();
			if (line==null)
			{
				//System.out.println("\nEnd of file\n");
				tokens2[0]="x";
			}
			else
				tokens2= line.split(delims);
		}

while(tokens[0].trim().equals(tokens2[0].trim())&&line!=null) ;

				elements[i]= Long.parseLong(tokens[0].trim());

					numbers.put(elements[i], i);

				TOTAL_NODES++;
```

```
                        countOfLinks[i]=count;
                        count=0;

            }


                  bufRead.close();
                  GetLinks();
                  CreateList();

             }
             catch (ArrayIndexOutOfBoundsException e)
             {
                  System.out.println("Usage: java ReadFile filename....\n");


             }
             catch (IOException e)
             {
                  // If another exception is generated, print a stack trace
            e.printStackTrace();
             }




   }

   public static void main(String[] args)
   {
         Parse_SN rd = new Parse_SN();

   }// end main

   public void GetLinks()
   {
         try
         {
               FileReader input = new FileReader("new.txt");

                  /* Filter FileReader through a Buffered read to read a line
at a
                     time */
               BufferedReader bufRead = new BufferedReader(input);


               String line;   // String that holds current file line
               String delims = "[,]";

               String[] tokens;
               tokens=new String[2];

               String[] tokens2;
               tokens2=new String[2];
               int i, j, flag1=0, flag2=0;

               line = bufRead.readLine();
```

```
                tokens= line.split(delims);

                for(i=0; line != null; i++)
                {
                      for(j=0; j<=countOfLinks[i]-1; j++)
                      {
                if(numbers.get(Long.parseLong(tokens[1].trim()))==null)
                            {
                                  elements[TOTAL_NODES]=
Long.parseLong(tokens[1].trim());

                                  numbers.put(elements[TOTAL_NODES],
TOTAL_NODES);

                                  TOTAL_NODES++;
                            }

                            int  l1=
Integer.parseInt(numbers.get(Long.parseLong(tokens[1].trim())).toString());

                            countOfLinks[l1]++;


                            line = bufRead.readLine();
                            flag1=0;
                            if(line!=null)
                                  tokens= line.split(delims);
                      }
                }

                System.out.print("Total number of nodes found (final): ");
                System.out.println(TOTAL_NODES);

                //Writer output = null;
              String str=new String("");
              File file = new File("write_links.txt");
              Writer output = new BufferedWriter(new FileWriter(file));
              for(i=0;i<TOTAL_NODES;i++)
              {
                str=str+i;

                str=str+": ";

                str=str+elements[i];
                str=str+": Count of Links: ";
                str=str+countOfLinks[i];
                str=str+";";
                str=str+"\n";

              }

              output.write(str);
              output.close();

              for(i=0;i<NUM_NODES;i++)
              {
                elements_links[i]= new long[countOfLinks[i]+1];
```

```
            }
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
              System.out.println("Usage: java ReadFile filename\n");

              e.printStackTrace();
        }
        catch (IOException e)
        {
              // If another exception is generated, print a stack trace
              e.printStackTrace();
        }



   }

   public void CreateList()
   {
        try
        {
              FileReader input = new FileReader("new.txt");

              /* Filter FileReader through a Buffered read to read a line at
a
                    time */
              BufferedReader bufRead = new BufferedReader(input);


          String line;       // String that holds current file line
          String delims = "[,]";

          String[] tokens;
          tokens=new String[2];

          String[] tokens2;
          tokens2=new String[2];

          String temp=new String("");
          String prnt=new String("");
          String output_str=new String("");
          int i, j, flag1=0, flag2=0, l1,l2,p=0;

          line = bufRead.readLine();
          tokens= line.split(delims);


          for(i=0; line!=null; i++)
          {
              temp=tokens[0];
              l1=
Integer.parseInt(numbers.get(Long.parseLong(tokens[0].trim())).toString());

              elements_links[l1][0]=Long.parseLong(tokens[0].trim());

              for(j=0; (j<=countOfLinks[l1]-1)&& (temp.equals(tokens[0]));
```

```
j++)
                {
                        if(Long.parseLong(tokens[1].trim())==0)
                        {
                                System.out.println("zero: "+line+ "temp: "+temp);
                        }
                        elements_links[l1][numOfLinks[l1]+1]  =
Long.parseLong(tokens[1].trim()); //(Long)numbers.get(tokens[1].trim());

                        numOfLinks[l1]++;

                        l2=
Integer.parseInt(numbers.get(Long.parseLong(tokens[1].trim())).toString());

                        //System.out.print("Printing l2 : ");
                        //System.out.println(l2);

                        elements_links[l2][0]=Long.parseLong(tokens[1].trim());


                        for(int k=1;(k<=countOfLinks[l2]-1)&&(flag1==0);k++)
                        {
                                if(elements_links[l2][k]==
Long.parseLong(tokens[0].trim()))
                                {
                                        flag1=1;
                                        break;
                                }
                        }
                        if(flag1==0)
                        {
                                p=numOfLinks[l2];
                                //System.out.print("p: ");
                                //System.out.println(p);
                                elements_links[l2][p+1]=
Long.parseLong(tokens[0].trim());
                                numOfLinks[l2]++;
                        }

                        line = bufRead.readLine();
                        flag1=0;
                        if(line!=null)
                        {
                                tokens= line.split(delims);
                        }

                } //end loop j

            } //end outer for loop i


            System.out.print("Total number of nodes found (final): ");
            System.out.println(TOTAL_NODES);

            Writer output = null;

            File file = new File("write_sn_5k.txt");
```

```
            output = new BufferedWriter(new FileWriter(file));

            for(i=0;i<TOTAL_NODES;i++)
            {
                output.write("create_nd(n");
                output.write(String.valueOf(elements_links[i][0]));

                output.write(", [");

                for(j=1;j<=numOfLinks[i]; j++)
                {

                      output.write("n");
                      int flg = 0;
                      for(int k=1;k<j; k++)
                      {
                            if(elements_links[i][j]==elements_links[i][k])
                            {
                                  flg = 1;
                                  break;
                            }
                      }
                      if(flg==1)
                            continue;
                      output.write(String.valueOf(elements_links[i][j]));

                      if(j!=numOfLinks[i])
                      {

                            output.write(",");
                      }
                }

                output.write("]),");
                output.write("\n");
            }

            output.write("List=[");
            for(i=0;i<TOTAL_NODES;i++)
            {
                output.write("n");

                output.write(String.valueOf(elements_links[i][0]));

                if(i!=TOTAL_NODES-1)
                {
                      //output_str=output_str+",";
                      output.write(",");
                }
            }

            output.write("],");
            output.write("\n");
            output.close();
          }
          catch (ArrayIndexOutOfBoundsException e)
          {
```

```
                System.out.println("Usage: java ReadFile filename\n");

                e.printStackTrace();
          }
          catch (IOException e)
          {
                // If another exception is generated, print a stack trace
            e.printStackTrace();
        }



    }


}
```

Figure A8: Java code for parsing graph and generating input for the simulation